



SISTEMI DIGITALI INTEGRATI

POLITECNICO DI TORINO

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI

Progetto Butterfly

Esercitazione finale di progetto sulle architetture integrate

Autori:

Palma Paolo 306046
Porcaro Mario 315888
Rinieri Filippo 317894

Professore:

Maurizio Zamboni

Gruppo 9

Anno accademico: 2022-2023

Contents

1	Derivazione del DFD	3
2	Timelife Diagram	9
3	Architettura dei bus	9
4	ROM rounder	12
5	Interfacciamento	13
6	Control Unit	17
7	Simulazioni	21
7.1	Butterfly	21
7.2	FFT	31
8	Appendice	34
8.1	pipe adder	34
8.2	pipe mult	35
8.3	ROM round	36
8.4	round and scale	37
8.5	late status PLA	39
8.6	uAddress Register	40
8.7	uInstruction Register	41
8.8	uROM	42
8.9	Contro Unit	43
8.10	Butterfly	46
8.11	FFT	53
8.12	TestButterfly.py	59
8.13	compile.do	64
8.14	FFT TEST	65

Abstract

In questa relazione viene presentato il progetto Butterfly, esercitazione finale del corso di Sistemi Digitali Integrati. Vengono motivate le scelte adottate in fase di progetto in base alle specifiche richieste e vengono descritti tutti i componenti necessari. Infine vengono presentate le simulazioni eseguite per verificare il funzionamento della macchina.

1 Derivazione del DFD

Si hanno a disposizione 3 tipi di operatori:

- Moltiplicatori
- Sommatori
- Blocchi per l'arrotondamento

I moltiplicatori sono in grado di effettuare due operazioni, selezionabili tramite un segnale di controllo:

- la moltiplicazione tra due dati, per la quale l'operatore ha un livello di pipe (il risultato sara' disponibile non nel ciclo corrente ma in quello successivo)
- la moltiplicazione di un dato per 2 agendo come uno shifter, per cui il risultato è immediatamente disponibile

Anche i sommatori sono in grado di effettuare due operazioni, selezionabili tramite un segnale di controllo, ovvero somme e sottrazioni. In entrambi i casi c'è un livello di pipe, per cui il risultato dell'operazione inviata al ciclo n sara' disponibile al ciclo n+1.

Infine vengono progettati dei blocchi (ROM) che effettuano l'arrotondamento, e rendono disponibile il risultato dopo un colpo di clock.

Si hanno a disposizione 1 moltiplicatore, 2 sommatori e un numero qualsiasi di blocchi per l'arrotondamento.

Per il momento si ipotizza di avere a disposizione risorse infinite per ottenere il limite superiore (upper bound) delle prestazioni ottenibili dalla nostra macchina. Il CDFD ottenuto in questo caso è il seguente:

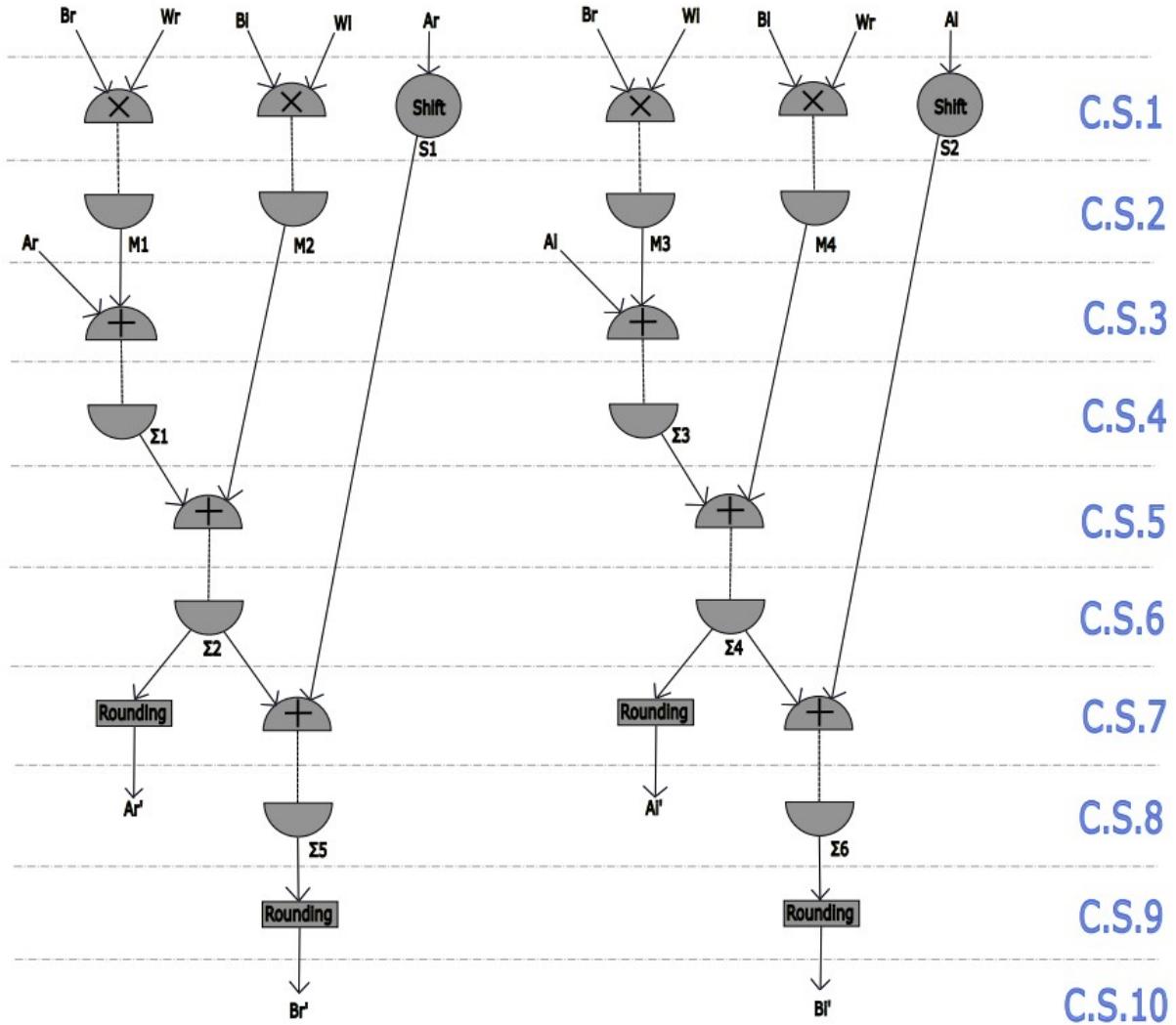


Figure 1: CDFD con risorse infinite

Per rappresentare il livello di pipe dei sommatori e dei moltiplicatori questi operatori sono stati divisi in due semicerchi. Gli ingressi vengono forniti all'inizio dello step algoritmico precedente e le relative uscite sono disponibili alla fine dello step algoritmico successivo. Il moltiplicatore in modalità shifter viene invece rappresentato con un cerchio intero dato che non è presente il livello di pipe. In questo caso è possibile completare le operazioni della Butterfly in 9 step usando 6 moltiplicatori, 2 sommatori e 2 operatori rounding. Si ha a disposizione un solo moltiplicatore, quindi è necessario fare dei miglioramenti. I 2 percorsi che necessariamente sono fissi a causa delle data dependencies sono:

$$M1 \rightarrow \Sigma_1 \rightarrow \Sigma_2 \rightarrow \Sigma_5 \quad (1)$$

$$M3 \rightarrow \Sigma_3 \rightarrow \Sigma_4 \rightarrow \Sigma_6 \quad (2)$$

Non potendo avere 2 moltiplicatori si deve necessariamente usare un control step in più e ciò ci permette di iniziare una moltiplicazione ad ogni step e poi fare in sequenza i 2 shift. Il nuovo CDFD ottenuto è:

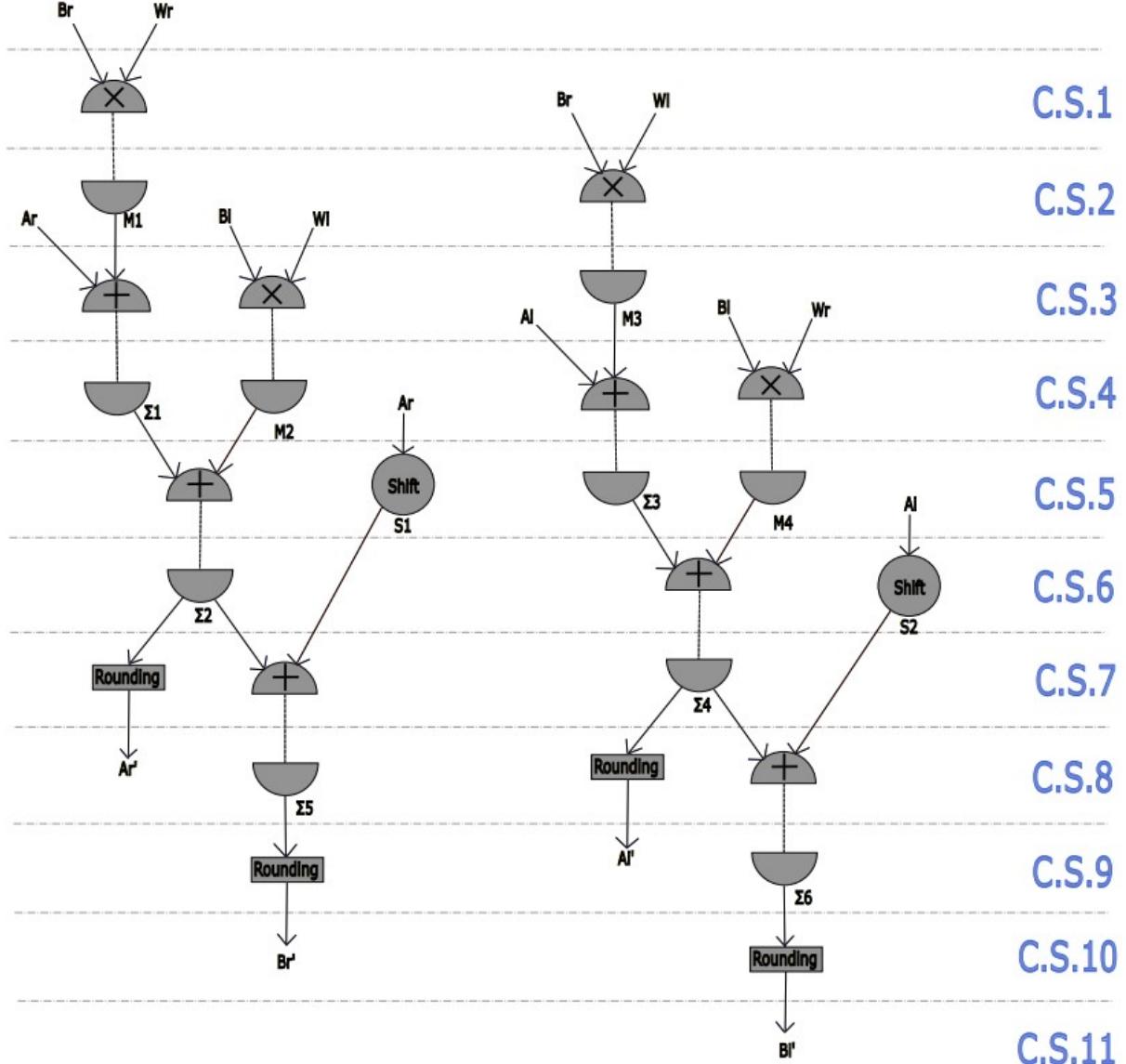


Figure 2: CDFD ASAP

Così si impiegano 10 control step, 1 moltiplicatore, 1 sommatore e 1 operatore di rounding. In questa struttura le uniche due operazioni che hanno grado di libertà sono gli shift, che possono essere fatti non appena il moltiplicatore si libera o un control step dopo. La soluzione in figura 2 è anche la soluzione ASAP, dato che non appena il moltiplicatore è disponibile viene subito fatto uno shift. In figura 3 viene riportata invece la soluzione ALAP, dove gli shift vengono fatti il più tardi possibile.

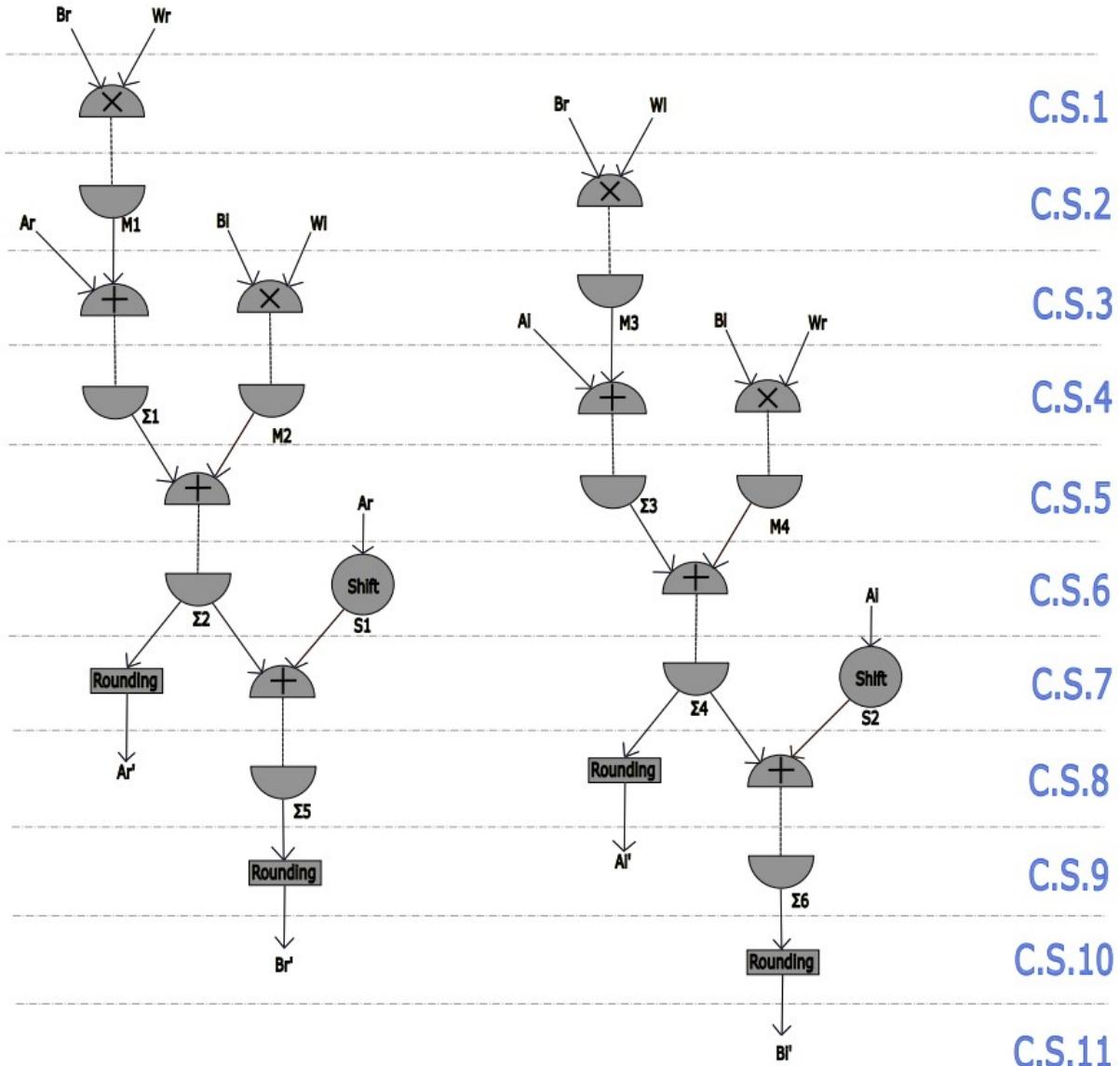


Figure 3: CDFD ALAP

In entrambe le soluzioni l'ultimo risultato è disponibile sempre dopo 10 control step. Il vantaggio della soluzione ALAP rispetto alla soluzione ASAP è che il risultato dello shift viene subito dato in ingresso al sommatore quindi non c'è bisogno di un registro in più per memorizzare $S1$ e $S2$.

In caso di esecuzione continua vogliamo che i calcoli relativi alla seconda esecuzione comincino il prima possibile, ovvero quando il moltiplicatore si libera. Abbiamo quindi due casi a seconda della scelta ASAP o ALAP:

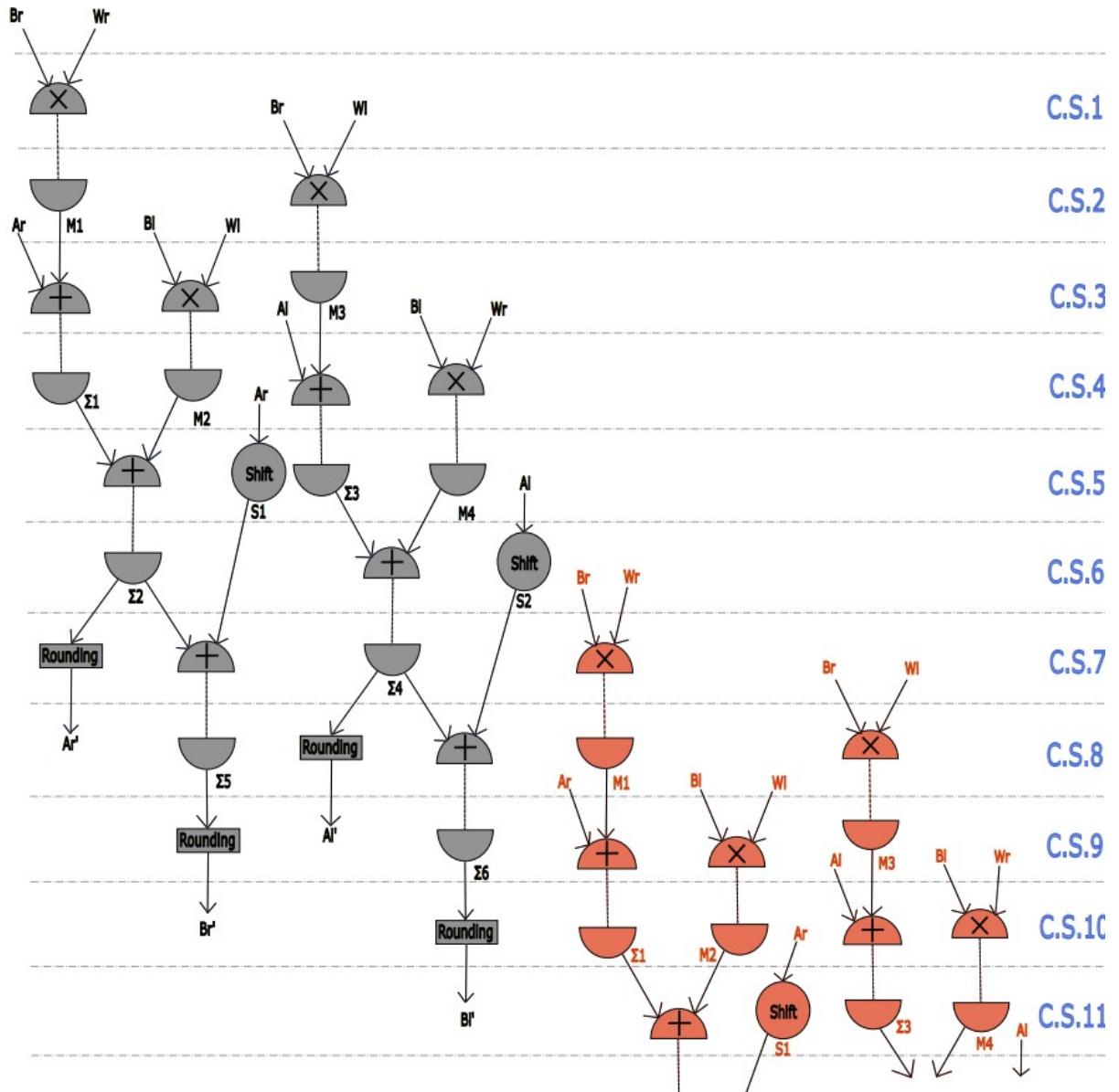


Figure 4: CDFD ASAP in esecuzione continua

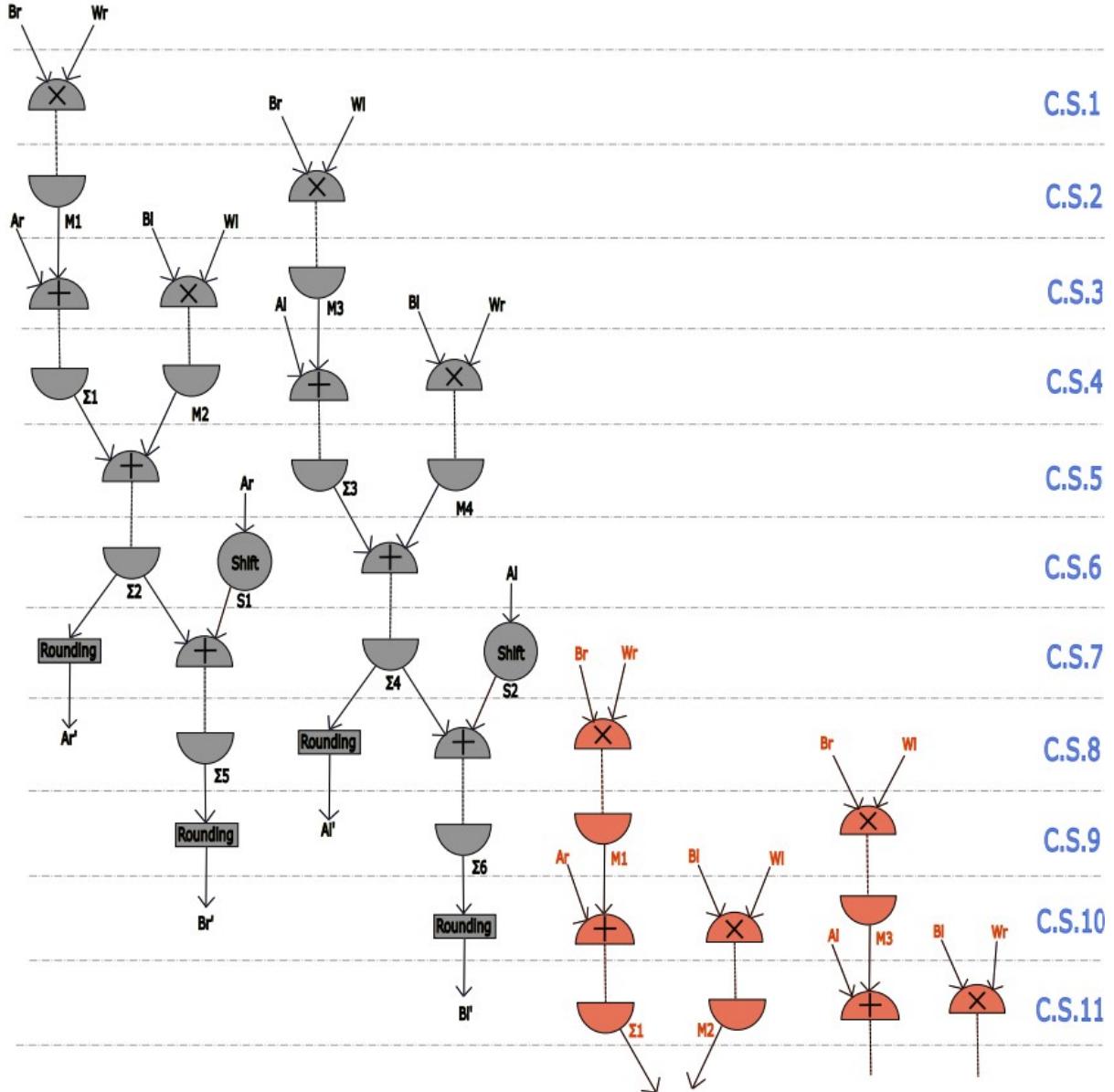


Figure 5: CDFD ALAP in esecuzione continua

Nel caso ASAP si riesce a iniziare la nuova esecuzione uno step prima, aumentando il Throghput. Per farlo però si introduce una complessità maggiore nel controllo: considerando che abbiamo bisogno di un colpo di clock per caricare i dati dall'esterno e di un colpo per renderli disponibili in uscita, in totale l'algoritmo ha una latenza di 13 colpi di clock. Adottando la soluzione ASAP si devono fornire nuovi ingressi ogni 6 colpi di clock. Ciò vuol dire che all'interno della Butterfly viaggeranno in contemporanea 3 set di ingressi, complicando il controllo. Si opta quindi per la soluzione ALAP, dove si forniscono nuovi ingressi ogni 7 colpi, possono viaggiare al massimo 2 set di ingressi in contemporanea e il controllo si semplifica.

2 Timelife Diagram

Per capire quanti registri servono per le variabili temporanee si ricorre al Timelife diagram, ovvero un diagramma temporale dove si dice per ogni registro per quanto tempo questo deve memorizzare l'informazione:

STEP	1	2	3	4	5	6	7	8	9	10	11
Ar		X			X						
Ai			X				X				
Br	X	X									
Bi			X	X							
Wr	X			X							
Wi		X	X								
M1			X								
M2					X						
M3			X								
M4					X						
S1						X					
S2							X				
$\Sigma 1$			X								
$\Sigma 2$						X					
$\Sigma 3$				X							
$\Sigma 4$							X				
$\Sigma 5$								X			
$\Sigma 6$									X		
TOTALE	0	0	1	1	2	2	2	2	1	1	0

Figure 6: Timelife Diagram

Le righe in giallo rappresentano il tempo di vita degli input, le righe in grigio il tempo di vita delle variabili temporanee. Siamo interessati ai registri che contengono le variabili temporanee, quindi per ciascuna colonna contiamo il numero di registri necessari e lo scriviamo nell'ultima riga. Il massimo numero scritto è 2, quindi servono 2 registri. Considerando che abbiamo 2 registri di uscita per il moltiplicatore (uno per il risultato della moltiplicazione e uno per il risultato dello shift) e un registro di uscita per il sommatore, si possono usare quelli anzichè aggiungerne di ulteriori.

3 Architettura dei bus

Avendo la necessità di far operare varie Butterfly su stadi successivi, abbiamo ritenuto opportuno dedicare uno stato della CU alla lettura simultanea di tutti gli input necessari all'elaborazione. Questa scelta si è rivelata oculata dal momento che, analizzando meglio l'algoritmo, possiamo notare come l'ordine dei risultati prodotti in uno stadio Butterfly precedente, non corrisponda all'ordine con cui questi dati verranno utilizzati nello stadio successivo. Risulta quindi necessario completare l'elaborazione dei quattro output e poi dare il segnale di DONE (che diventerà lo START degli stadi successivi).

Dovendo quindi memorizzare sei valori in ingresso (INPUT + coefficienti) e quattro valori in uscita, è stato necessario utilizzare dieci registri per ogni Butterfly. Abbiamo preferito per semplicità, dato il numero contenuto di registri, optare per una distribuzione di "registri sparsi" all'interno del datapath. Possiamo vederli raffigurati in figura 7, incolonnati per semplicità.

DATA IN:
 +DA UTENTE O STADI BUTTERFLY PREC.
 +DA BUS COEFFICIENTI

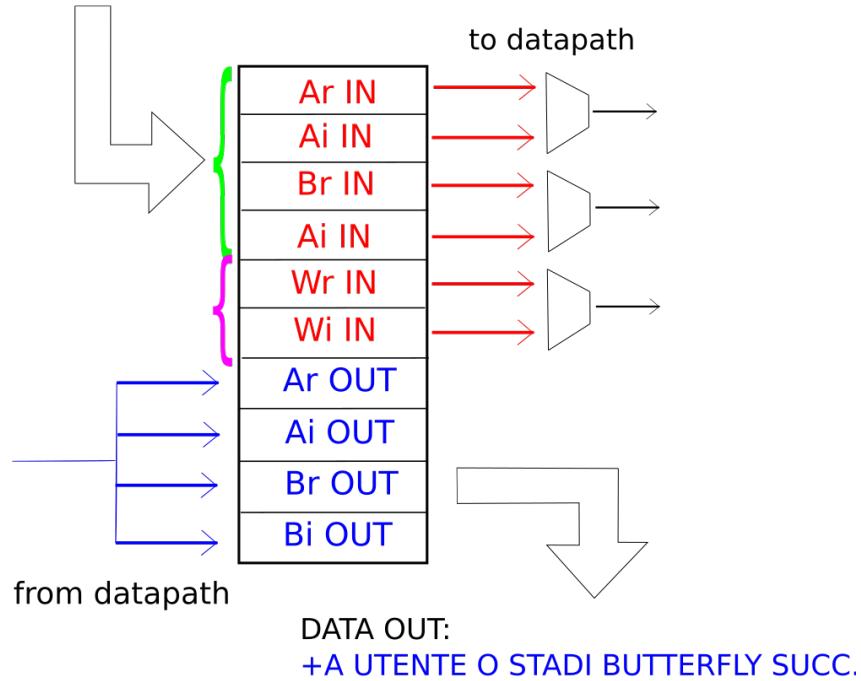


Figure 7: Architettura dei bus

Dall'immagine 7 notiamo come tutti gli input necessari all'elaborazione della Butterfly passino per opportuni MUX (si veda datapath completo) che ci permette di ridurre il più possibile la dimensione dei collegamenti interni. Allo stesso modo notiamo come i dati provenienti dal datapath viaggino su un unico bus fino al momento dello scalamento finale, ma poi vengano successivamente inviati su quattro diverse linee dati (ciascuna per ogni registro di OUTPUT -in blu-); ovviamente verrà campionato solo il registro opportuno.

Per quanto riguarda l'interfacciamento con l'utente (o stadi successivi/precedenti di Butterfly) possiamo notare come i coefficienti utilizzati dall'algoritmo non vengano prodotti in fase di elaborazione, ma forniti esternamente dall'utente. Potrebbe quindi essere sensato, nell'ottica del funzionamento a stadi per il calcolo di una FFT, prevedere un BUS ad hoc per i vari Wr e Wi condivisi da vari stadi per evitare ridondanza.

Il datapath della butterfly così progettata è mostrato in figura 8.

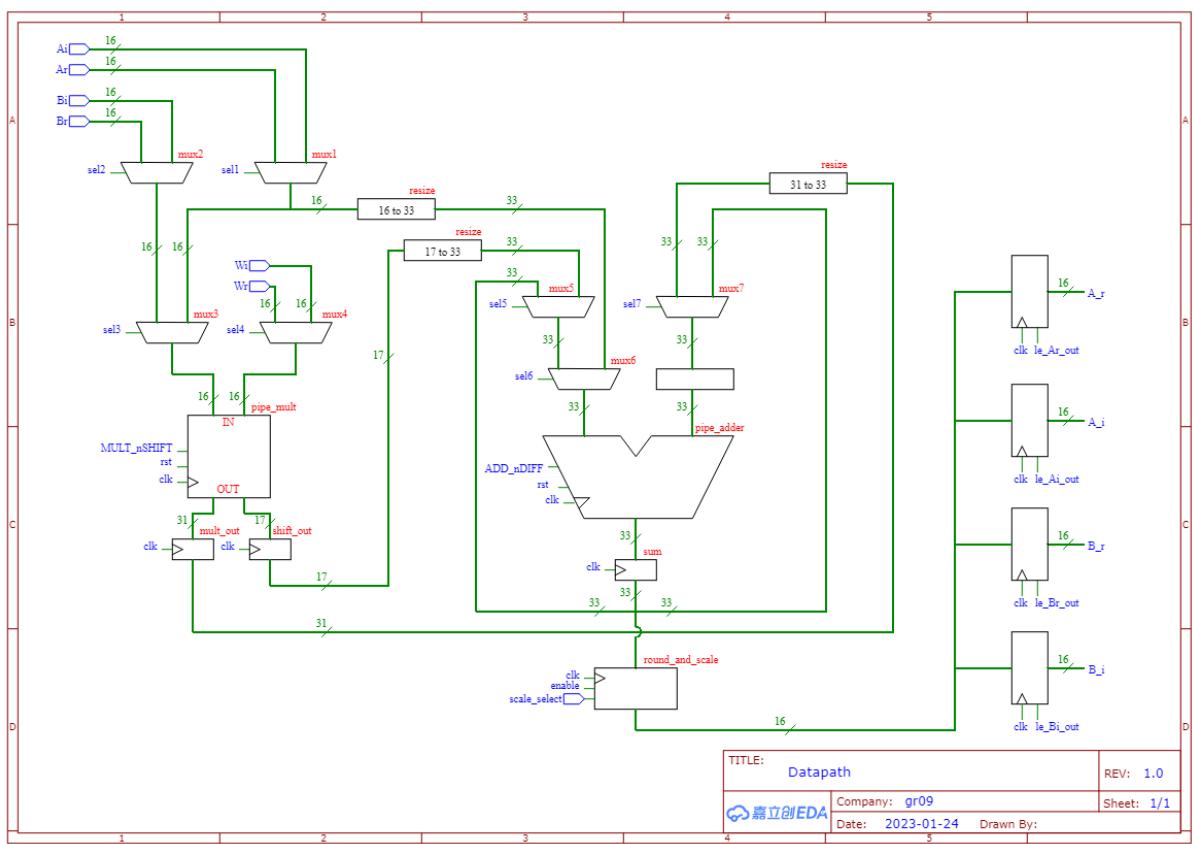


Figure 8: Datapath

4 ROM rounder

Alla fine delle elaborazioni funzionali al calcolo di ogni output, sarà necessario arrotondare il risultato ottenuto per rientrare sui 16 bit previsti in uscita.

Da traccia ci è stato imposto l'utilizzo della tecnica del “ROM rounding” che, in breve, ci permette di arrotondare gli output ottenuti al costo di un accesso in memoria, piuttosto che arrotondare esplicitamente per mezzo di operazioni matematiche.

In particolare possiamo osservarne il funzionamento in figura 9:

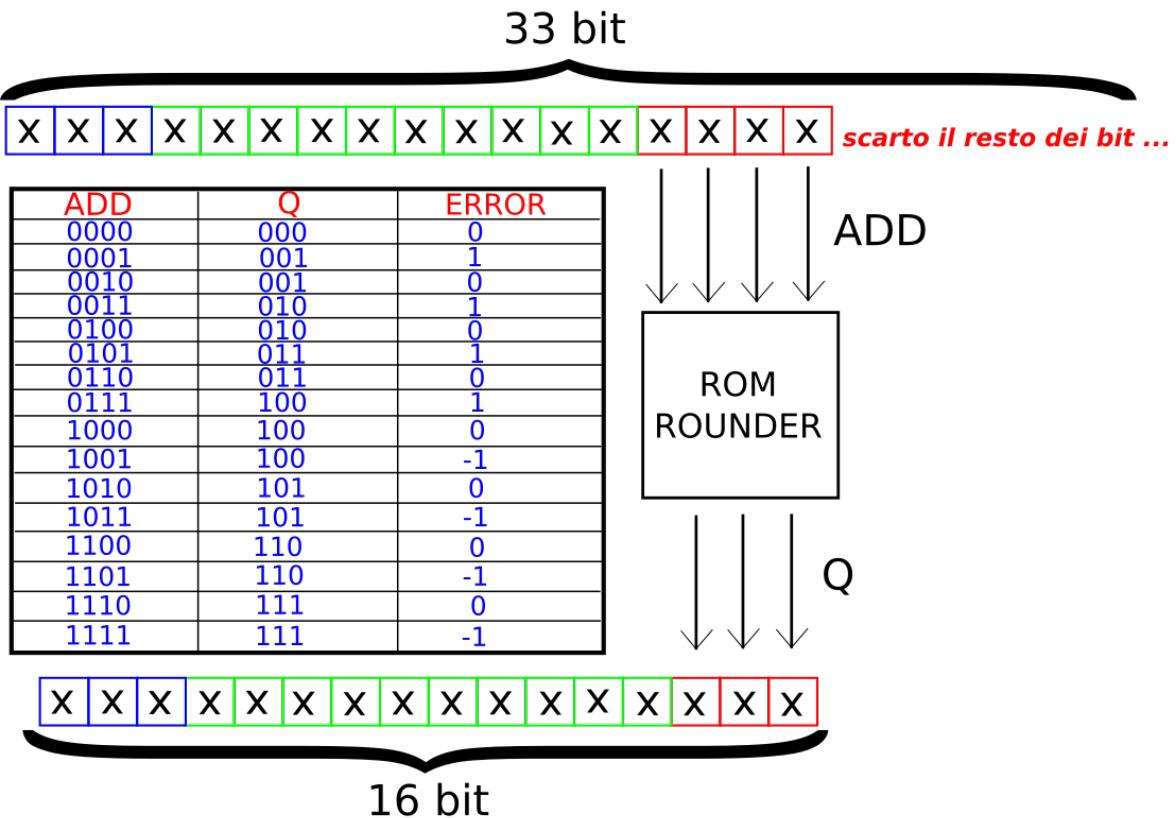


Figure 9: Implementazione ROM rounding

Possiamo notare come, agendo opportunatamente sulla relazione che lega ADD (indirizzi ROM) e Q (uscite ROM), siamo stati in grado di rendere l'errore medio commesso in fase di arrotondamento pari a 0.

In tabella sono stati utilizzati dei valori interi per semplicità; volendo essere rigorosi al posto di ± 1 dovrebbe esserci $\pm 2^{-14}$.

5 Interfacciamento

La butterfly deve campionare 7 segnali in ingresso e fornire 4 segnali di dato di uscita:

- Ingressi: $A_r, A_i, B_r, B_i, W_r, W_i, \text{SF_1h_01}$
- Uscite: $A_r\text{Out}, A_i\text{Out}, B_r\text{Out}, B_i\text{Out}$

Un segnale di START viene dato dall'esterno alla Butterfly per farla partire, un segnale di DONE viene fornito dalla Butterfly per segnalare che un'elaborazione è terminata e i dati in uscita sono validi. Per realizzare la FFT più Butterfly andranno collegate in successione, per cui l'uscita deve essere interfacciabile con l'ingresso.

L'idea più semplice e "ordinata" è la seguente: campionare tutti gli ingressi nello stesso istante e fornire tutte le uscite nello stesso istante. Ciò vuol dire che, collegando due Butterfly in sequenza, si aspettano che i 4 segnali siano tutti nei rispettivi registri e poi si forniscono verso l'esterno. Alcune di queste uscite erano però già disponibili da prima, ad esempio $A_r\text{Out}$ è la prima ad essere generata e scritta nel suo registro. Non si potrebbe mandarla alla prossima Butterfly senza aspettare le altre uscite? La Butterfly successiva non potrebbe comunque partire, perché l'ordine con cui le uscite vengono generate è diverso dall'ordine con cui vengono usati gli ingressi. Ad esempio il primo ingresso ad essere usato nell'algoritmo è B_r , ma il suo corrispettivo $B_r\text{Out}$ non è il primo segnale ad uscire, bensì il terzo.

Per cui si sceglie il seguente interfacciamento: il segnale START viene asserito e nel colpo di clock successivo vengono campionati contemporaneamente i 7 segnali di ingresso. Dopo 13 colpi di clock viene asserito il segnale DONE, ciò vuol dire che le uscite sono stabili e possono essere prelevate. Questo è il caso più semplice di esecuzione "singola" della Butterfly:

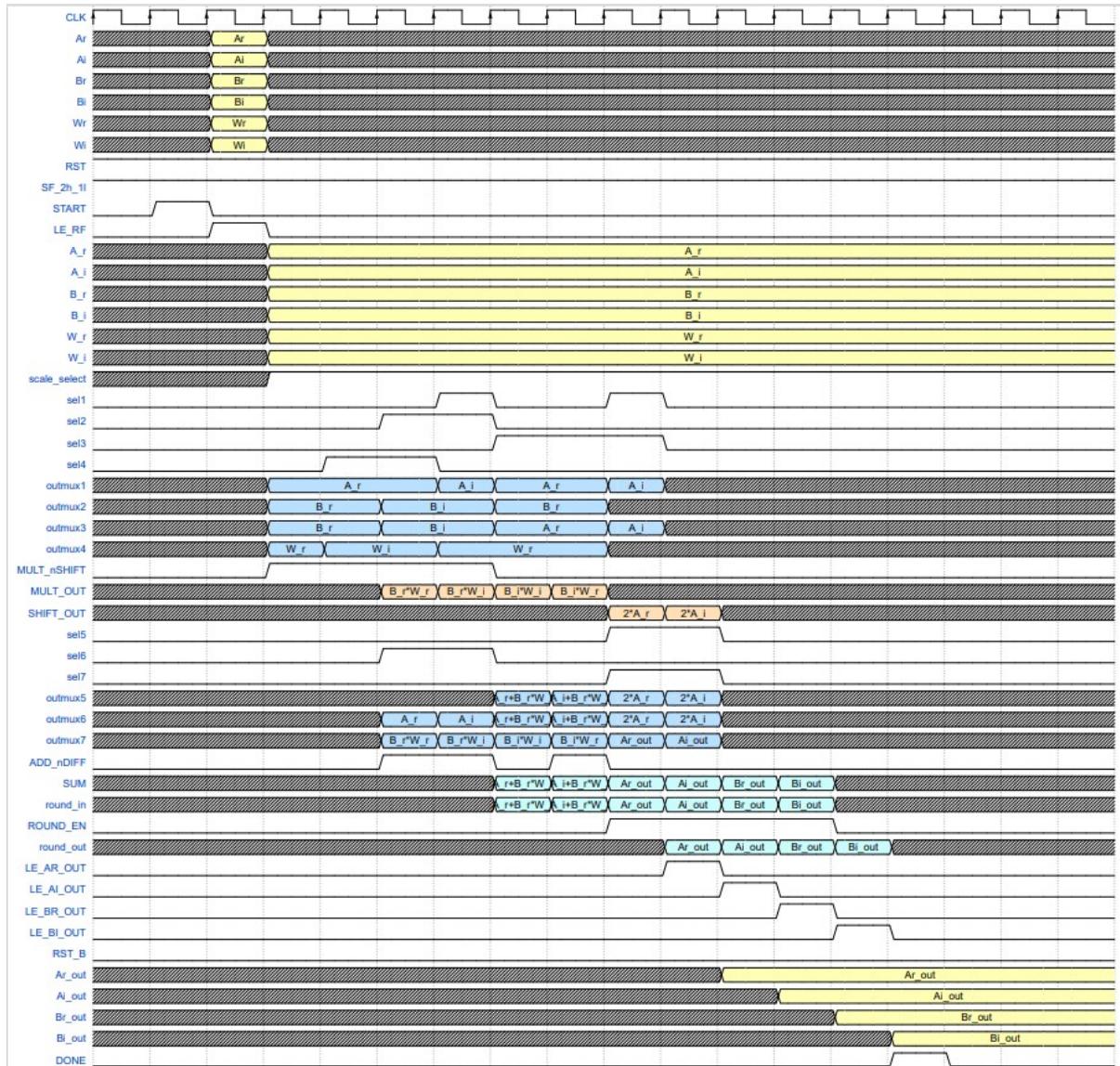


Figure 10: Timing Diagram per in modalità singola

Per far sì che la macchina lavori in modalità continua basta asserire il segnale START dopo 7 colpi di clock dall'esecuzione precedente e fornire i nuovi input al colpo di clock successivo così che possano essere campionati. Dopo 13 colpi di clock avremo gli output corrispondenti. Il timing diagram dell'esecuzione continua è il seguente:

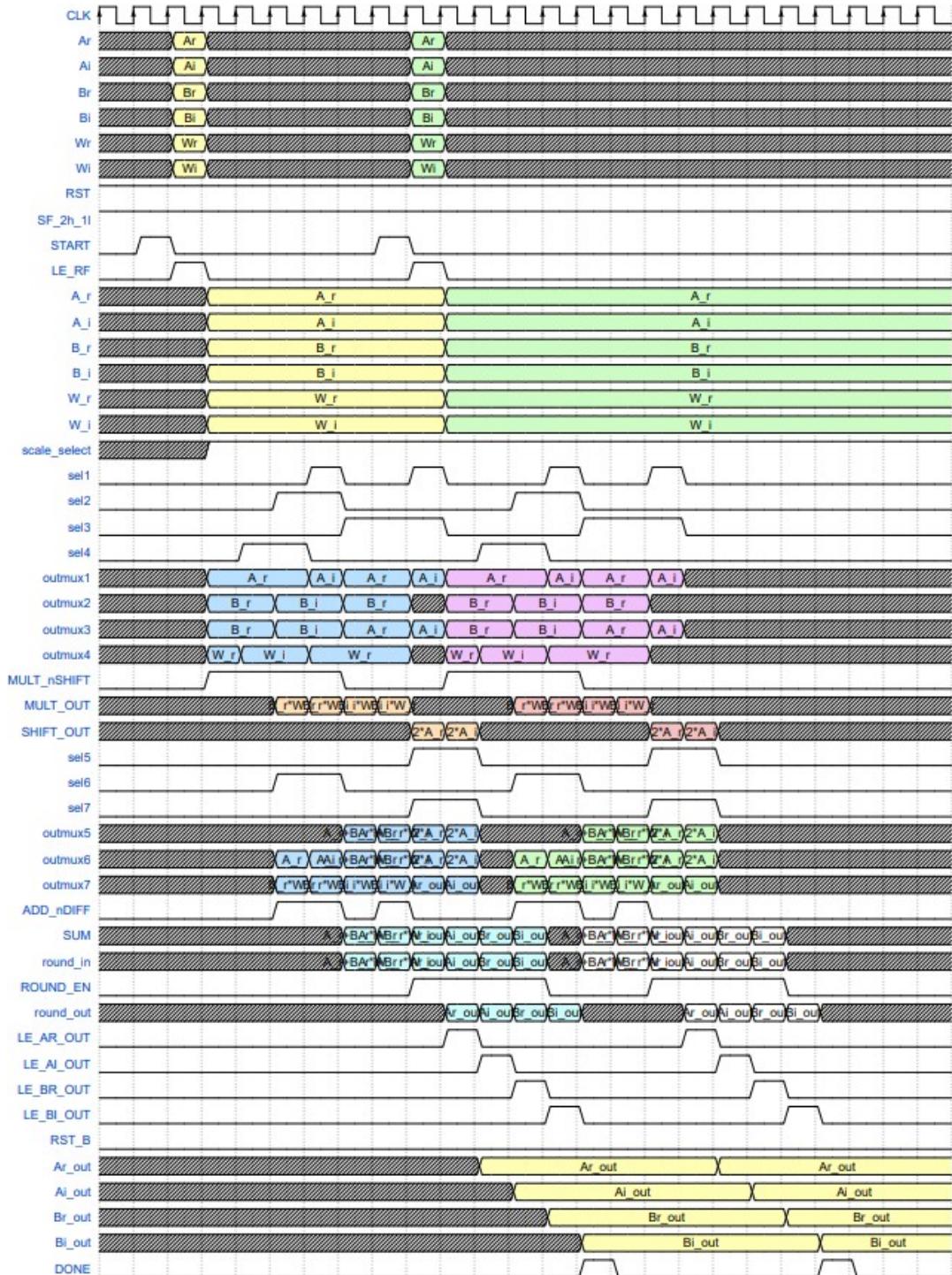


Figure 11: Timing Diagram per in modalità continua (in questo caso solo 2 esecuzioni in continua)

Da questo timing diagram si ricava facilmente la tabella degli stati e la macchina a stati della Butterfly:

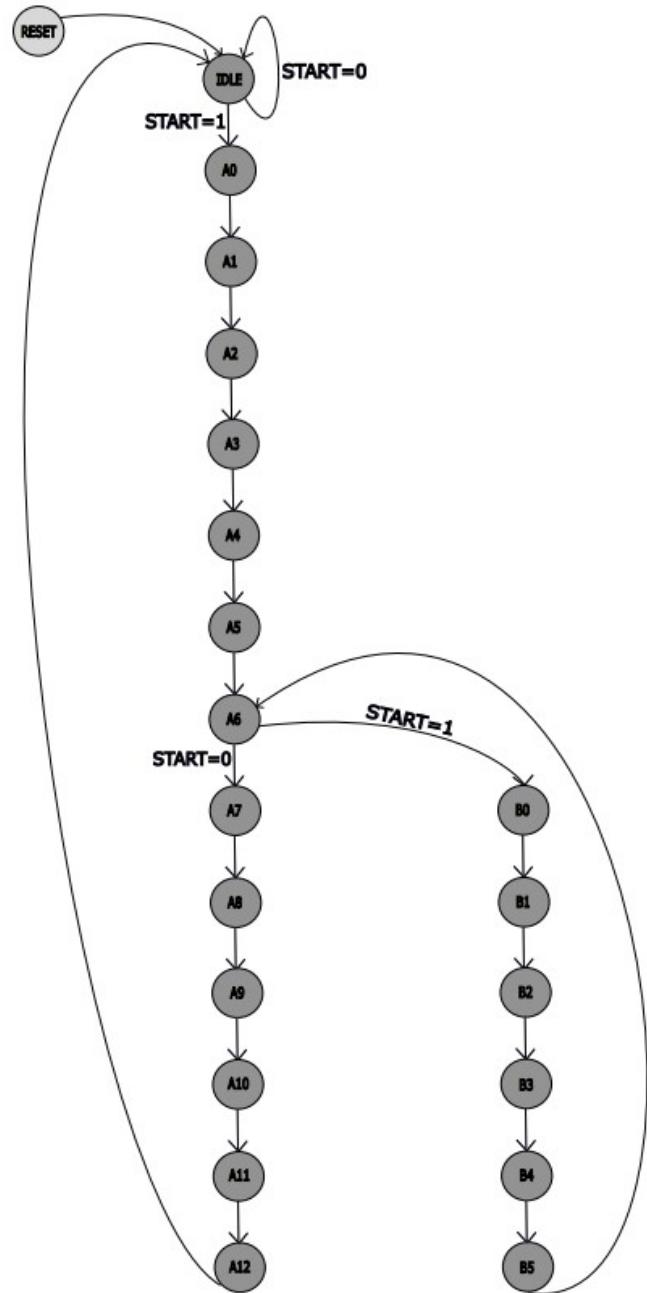


Figure 12: FSM

STATO	LE_RF	MUX1	MUX2	MUX3	MUX4	MUX5	MUX6	MUX7	Mul/Shift	Sum/Diff	ROM roun	LE Ar ⁱ	LE Ai ⁱ	LE Br ⁱ	LE Bi ⁱ	Done	RST
Idle	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
A2	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
A3	0	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0
A4	0	1	1	0	0	0	1	0	1	1	0	0	0	0	0	0	0
A5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
A6	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
A7	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0
A8	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0
A9	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
A10	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0
A11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
A12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
B0	1	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0
B1	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0
B2	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0
B3	0	0	1	0	1	0	1	0	1	1	1	0	0	1	0	0	0
B4	0	1	1	0	0	0	1	0	1	1	0	0	0	0	1	0	0
B5	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 13: Tabella degli stati

6 Control Unit

Progettato completamente il datapath, si passa a definire l'unità di controllo. Si usa un sequenziatore con indirizzamento esplicito e con la tecnica del LATE STATUS: la microrom è strutturata in modo tale che abbia metà delle righe rispetto al , e dove ogni riga è fatta da una cella pari e da una cella dispari. Abbiamo 21 stati, normalmente sarebbero necessari 5 bit per puntare alla singola cella che contiene un singolo stato; nel caso LATE STATUS si prendono dal Next Address tutti i bit tranne l'ultimo, quindi 4 bit, per tirare fuori dalla uROM una riga contenente 2 stati. A questo punto, è chiaro che se avremo due dati, dovremo selezionarne uno dei due; quindi se andiamo in sequenza, è 0 quello pari, è 1 quello dispari. In caso di possibili salti (segnalati dal bit CC che vale 1), la Late Status PLA può andare ad agire sul solito MUX ad un bit per selezionare lo stato corretto anche in base ai segnali di controllo provenienti Datapath o dall'esterno.

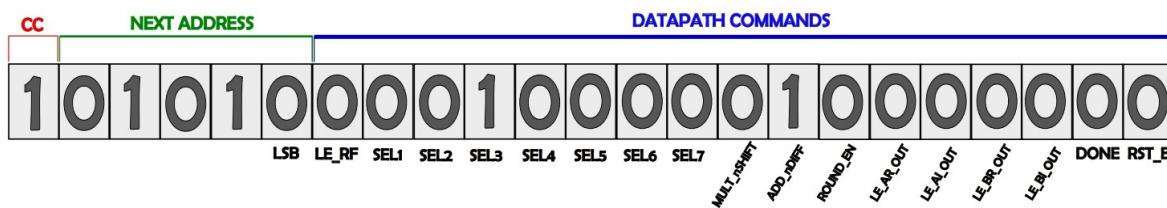


Figure 14: Contenuto del μ IR

In Figura 14 viene riportato il contenuto del microInstructionRegister:

- CC: il Condition Code permette di capire se avviene un salto condizionato (CC=1) o se si procede in sequenza
- NEXT ADDRESS: il next address codificato è codificato su 5 bit, i primi 4 saranno direttamente copiati nel micro Address Register mentre l'LSB entrerà nella Late Status PLA

- Segnali di controllo: dato che il numero di segnali di controllo non è eccessivamente alto (17), si decide di non fare alcun tipo di codifica. Ad ognuno dei 17 bit di controllo corrisponde un segnale di controllo (Horizontal uProgramming)

microAR	PresentState	NextState	CC	NS	microIR	
00000	RESET	IDLE	0	00010	00000000000000000001	
00001						
00010	IDLE	IDLE/A0	1	00010	00000000000000000000	
00011	A0	A1	0	00100	10000000000000000000	
00100	A1	A2	0	00101	00000000100000000000	
00101	A2	A3	0	00110	00001000100000000000	
00110	A3	A4	0	00111	00101010110000000000	
00111	A4	A5	0	01000	01100010110000000000	
01000	A5	A6	0	01001	00010000000000000000	
01001	A6	A7/B0	1	01010	00010000010000000000	
01010	A7	A8	0	01100	01010101001000000000	
01011	B0	B1	0	10001	11010101001000000000	
01100	A8	A9	0	01101	00000101001100000000	
01101	A9	A10	0	01110	00000000000101000000	
01110	A10	A11	0	01111	00000000000100100000	
01111	A11	A12	0	10000	00000000000000001000	
10000	A12	IDLE	0	00010	00000000000000000010	
10001	B1	B2	0	10010	00000101101100000000	
10010	B2	B3	0	10011	00001000101010000000	
10011	B3	B4	0	10100	00101010110010000000	
10100	B4	B5	0	10101	01100010110000100000	
10101	B5	A6	0	01010	00010000000000001000	
10110						
10111						
11000						
11001						
11010						
11011						
11100						
11101						
11110						
11111						

Figure 15: Codifica degli stati

In Figura 15 vengono codificati tutti gli stati. Lo stato di RESET corrisponde alla codifica 00000 in modo da rendere più semplice il reset asincrono del uAR. Per poter far funzionare il Late Status è necessario che gli indirizzi a valle di un possibile salto siano uguali tranne per l'LSB. Ci sono due possibili salti: In Figura 14 viene riportato il contenuto del microInstructionRegister:

- da IDLE si può saltare in IDLE oppure A0
- da A6 si può saltare in A7 oppure B0

Quindi gli indirizzi corrispondenti alle coppie IDLE-A0 e A7-B0 devono differire solo dell'LSB.

Per quanto riguarda gli stati non codificati questi non possono essere lasciati al caso: all'avvio o per colpa di un qualsiasi disturbo la macchina a stati potrebbe trovarsi in uno stato non previsto e bloccarsi. Per questo in corrispondenza di stati non codificati si inserisce la codifica corrispondente alla stato RESET. In caso di problemi la macchina si resetterà e al colpo di clock successivo si troverà in stato IDLE, in attesa dello START. La uROM sarà quindi organizzata così:

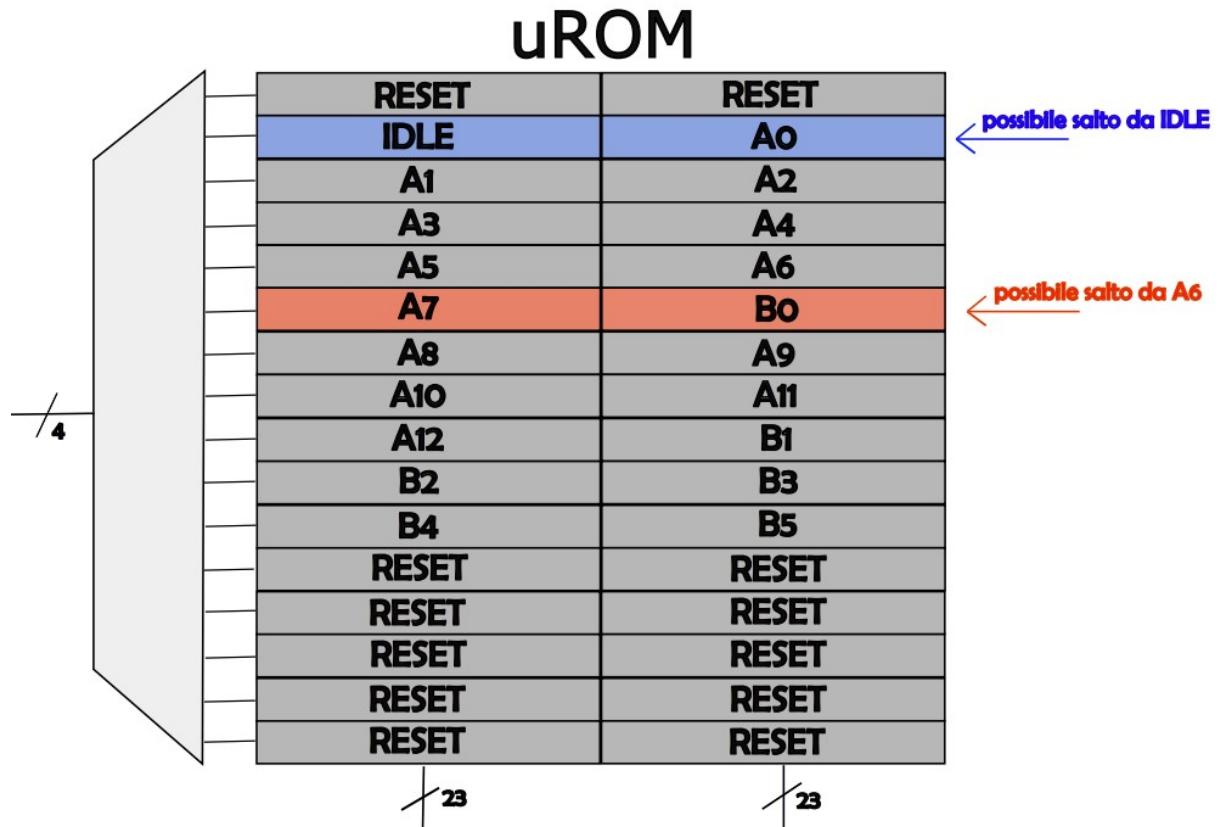


Figure 16: Contenuto della μ ROM

In caso di possibile salto la LATE STATUS PLA decide cosa fare in base al valore dei segnali di stato. In questo caso non si sono segnali di stato provenienti direttamente dal Datapath, ma la decisione dipende unicamente dal segnale START per entrambi i possibili salti.

La Control Unit così ottenuta è la seguente:

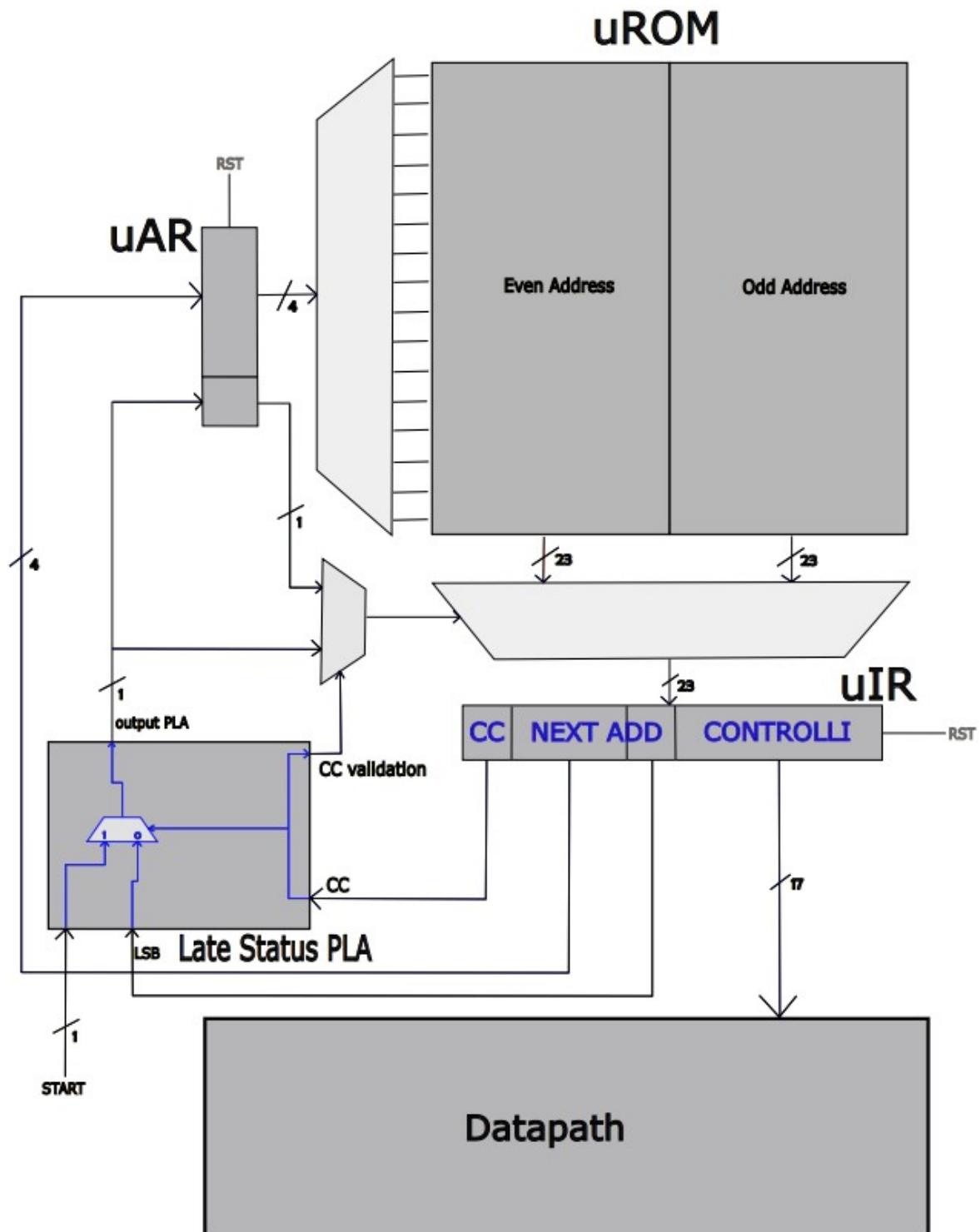


Figure 17: CU della Butterfly (Late Status)

E' presente un reset asincrono sia per il uAR che per il uIR:

Senza il reset asincrono del uIR ci sarebbe un delay tra l'asserzione del RST e l'effettivo reset della macchina: il valore 00000 viene scritto nel uAR e dopo un tempo di accesso T_{acc} il valore corrispondente dei nuovi segnali controllo viene scritto nel uIR. Resetando contemporaneamente entrambi i registri si cambiano i segnali di controllo istantaneamente.

7 Simulazioni

7.1 Butterfly

Per testare l'architettura sviluppata per implementare la nostra Butterfly abbiamo utilizzato uno script Python (consultabile in appendice). In questo script andiamo come prima cosa a generare alcuni file contenenti una serie di input randomici che verranno poi dati contemporaneamente in pasto a:

- un algoritmo scritto in Python per simulare il comportamento della Butterfly, col fine di avere quattro vettori di output corretti (Ar_{out} , Ai_{out} , Br_{out} , Bi_{out}) con cui confrontare gli output della nostra architettura VHDL;
- l'architettura VHDL da noi sviluppata, lo script Python è infatti in grado di invocare l'esecuzione di ModelSim parametrizzata dal file compile.do (consultabile anch'esso in appendice).

Il file compile.do si occupa di compilare tutti i *components* necessari alla definizione della nostra architettura e di avviare la simulazione del testbench per il tempo desiderato, oltre che chiudere il programma conclusa la simulazione. Il testbench, d'altro canto, è scritto in maniera tale da “campionare” in maniera opportuna gli input generati dal Python e gli output in uscita dall'entity sotto esame, andandoli ad inserire in opportuni file .txt.

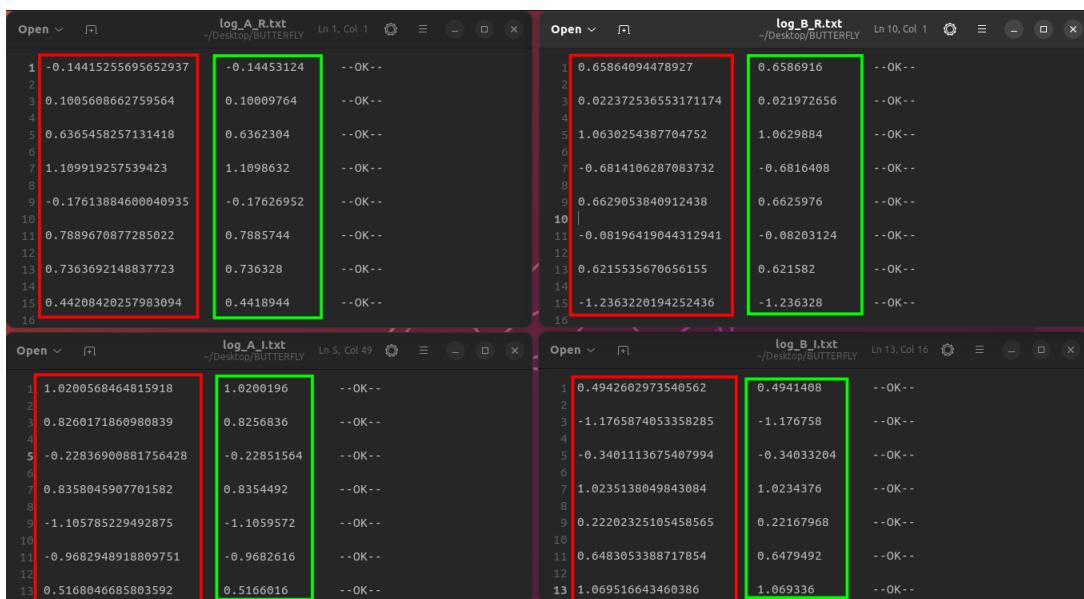
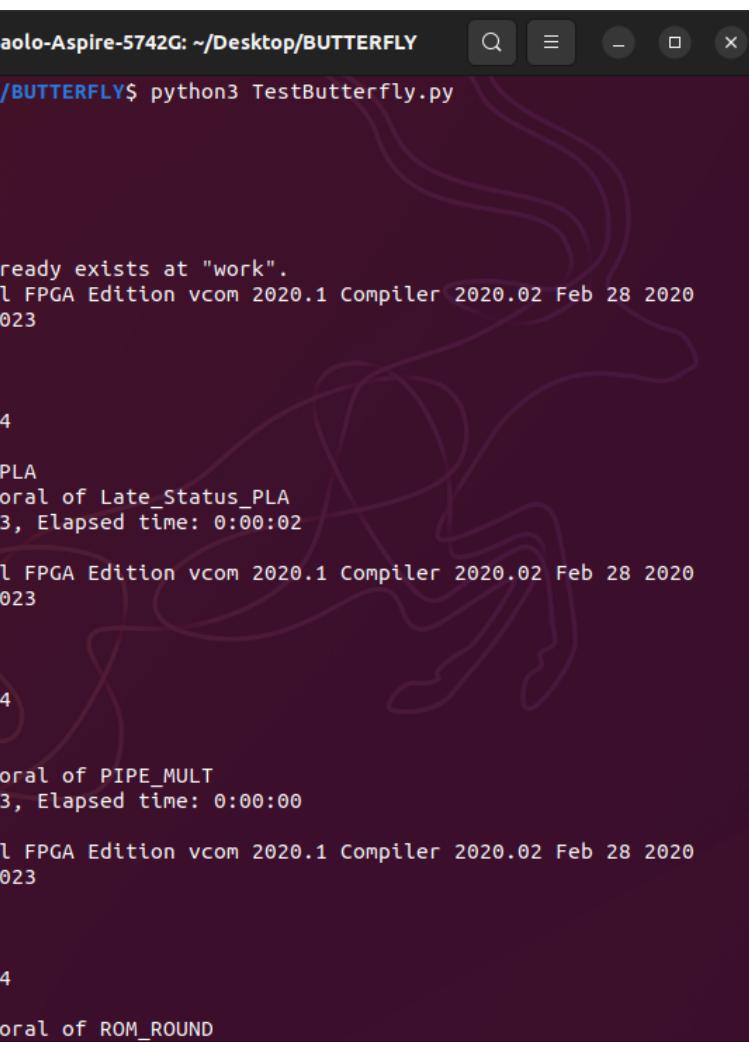


Figure 18: Confronto dei file di log delle simulazioni

Lo script si occupa inoltre di generare quattro diversi file di log (per ciascuno dei vettori di output) nei quali è possibile osservare i due output in esame affiancati e seguiti da un commento: “OK” oppure “ERROR”, come mostrato in figura 18. In rosso abbiamo i risultati dell’algoritmo Python, in verde gli output del VHDL.

Tutti i risultati presentati di seguito fanno riferimento a simulazioni dell’architettura con due shift finali, è stata testata anche la versione con singolo shift che si è dimostrata funzionare analogamente.

Andando a confrontare opportunatamente le coppie di vettori di output ottenuti, col fine di calcolare l’errore commesso (in particolare come valore assoluto della loro differenza), lo script segnalerà con “ERROR” ogni coppia di output che genera un errore superiore ad una soglia impostata (in questo caso 2^{-11} tenendo conto degli errori di arrotondamento e della quantizzazione dei numeri randomici prodotti dallo script Python); il numero di errori totali riscontrati verrà inoltre utilizzato per calcolare la percentuale di errore sul totale degli output analizzati.



```

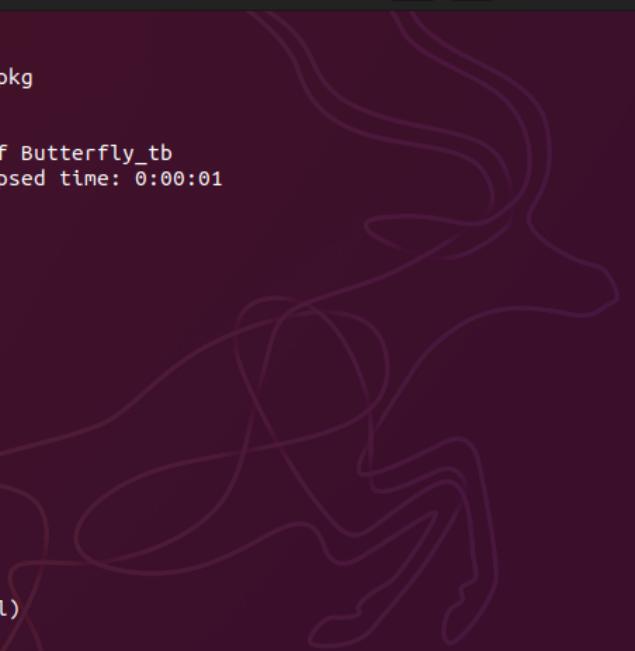
paolo@paolo-Aspire-5742G:~/Desktop/BUTTERFLY$ python3 TestButterfly.py
Starting simulation...
Reading pref.tcl

# 2020.1

# do compile.do
# ** Warning: (vlib-34) Library already exists at "work".
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 20:16:51 on Feb 11,2023
# vcom Late_Status_PLA.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Compiling entity Late_Status_PLA
# -- Compiling architecture behavioral of Late_Status_PLA
# End time: 20:16:53 on Feb 11,2023, Elapsed time: 0:00:02
# Errors: 0, Warnings: 0
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 20:16:53 on Feb 11,2023
# vcom PIPE_MULT.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package NUMERIC_STD
# -- Compiling entity PIPE_MULT
# -- Compiling architecture behavioral of PIPE_MULT
# End time: 20:16:53 on Feb 11,2023, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# Model Technology ModelSim - Intel FPGA Edition vcom 2020.1 Compiler 2020.02 Feb 28 2020
# Start time: 20:16:53 on Feb 11,2023
# vcom ROM_ROUND.vhd
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Compiling entity ROM_ROUND
# -- Compiling architecture behavioral of ROM_ROUND

```

Figure 19: Schermata del terminale



```
# -- Loading package fixed_generic_pkg
# -- Loading package MATH_REAL
# -- Loading package body fixed_generic_pkg
# -- Loading package std_logic_textio
# -- Compiling entity Butterfly_tb
# -- Compiling architecture behavioral of Butterfly_tb
# End time: 20:16:55 on Feb 11,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
# vsim -c work.Butterfly_tb
# Start time: 20:16:55 on Feb 11,2023
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading ieee.fixed_float_types
# Loading ieee.math_real(body)
# Loading ieee.fixed_generic_pkg(body)
# Loading ieee.fixed_pkg
# Loading ieee.std_logic_textio(body)
# Loading work.butterfly_tb(behavioral)
# Loading work.butterfly(behavioral)
# Loading work.pipe_mult(behavioral)
# Loading work.pipe_adder(behavioral)
# Loading work.round_and_scale(behavioral)
# Loading work.rom_round(behavioral)
# Loading work.cu(behavioral)
# Loading work.late_status_pla(behavioral)
# Loading work.urom(behavioral)
# Loading work.uaddress_register(behavioral)
# Loading work.uinstruction_register(behavioral)
# End time: 20:16:57 on Feb 11,2023, Elapsed time: 0:00:02
# Errors: 0, Warnings: 0
Simulation completed
Percentuale output Ar con errore > 2^-11 = 0.000 %
Percentuale output Ai con errore > 2^-11 = 0.000 %
Percentuale output Br con errore > 2^-11 = 0.000 %
Percentuale output Bi con errore > 2^-11 = 0.505 %
paolo@paolo-Aspire-5742G:~/Desktop/BUTTERFLY$
```

Figure 20: Schermata del terminale

Infine, con l'ausilio della libreria matplotlib di Python, andiamo a graficare i risultati ottenuti. In particolare avremo:

Confronto tra output ottenuti da Python e output ottenuti da ModelSim, mostrato nelle figure 21, 22, 23 e 24.

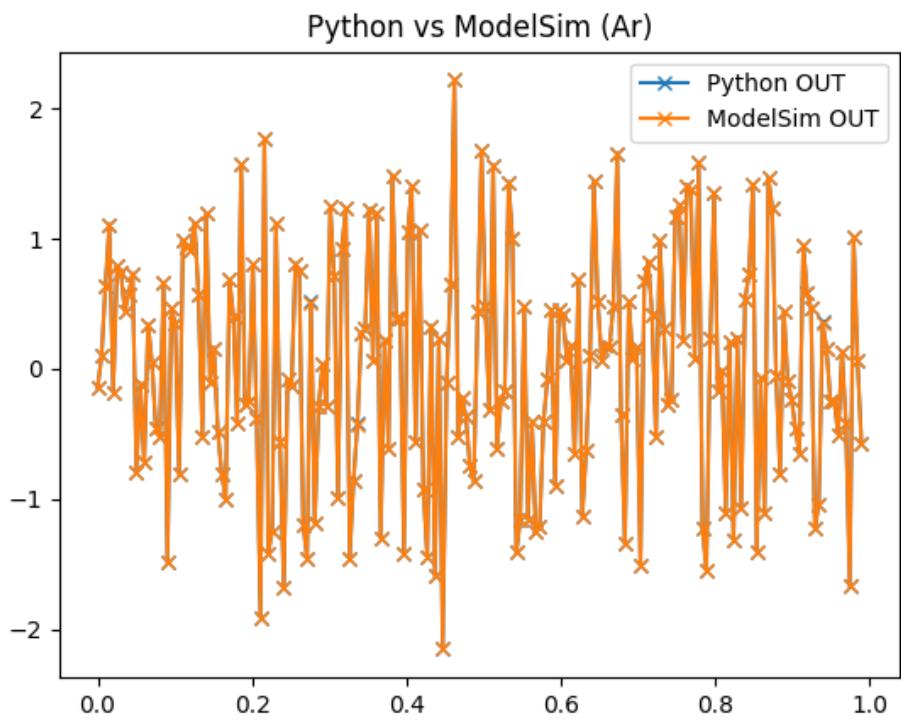


Figure 21: Simulazione Python e ModelSim per il calcolo di Ar_out

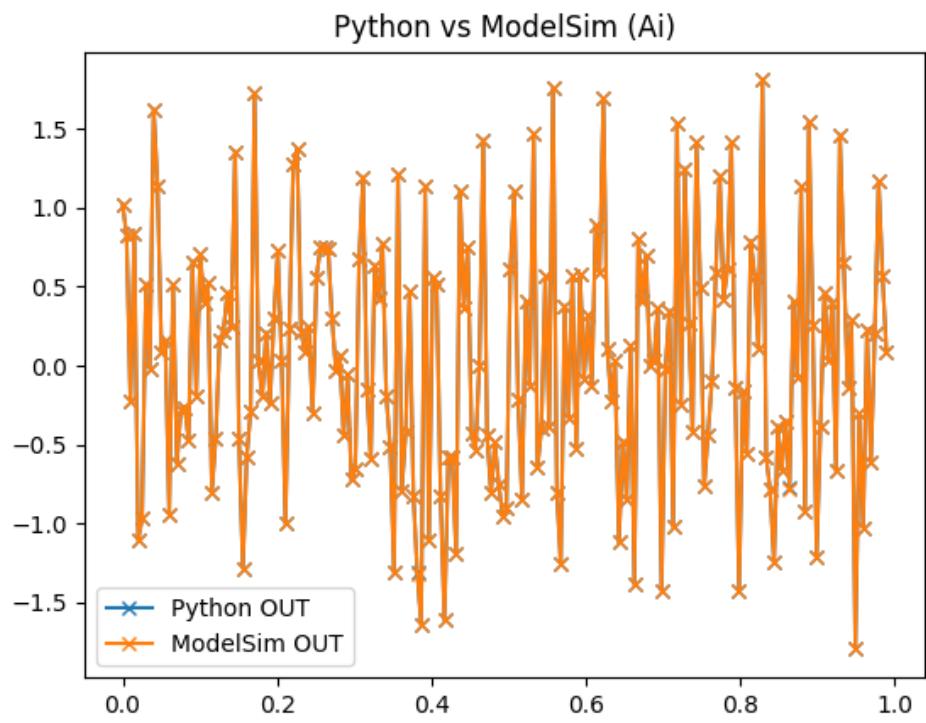


Figure 22: Simulazione Python e ModelSim per il calcolo di Ai_out

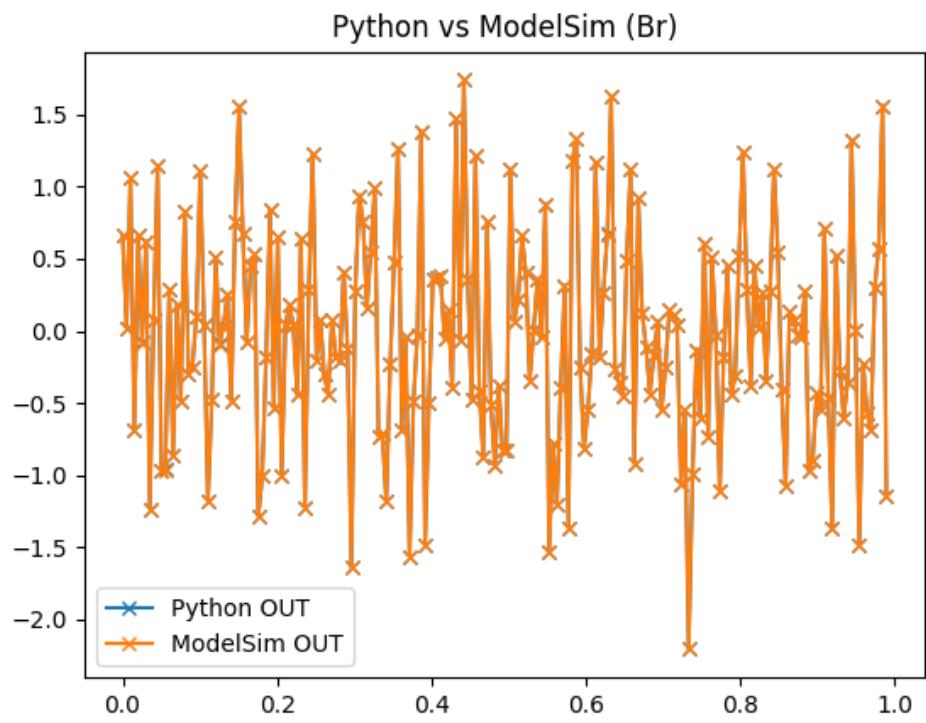


Figure 23: Simulazione Python e ModelSim per il calcolo di Br_out

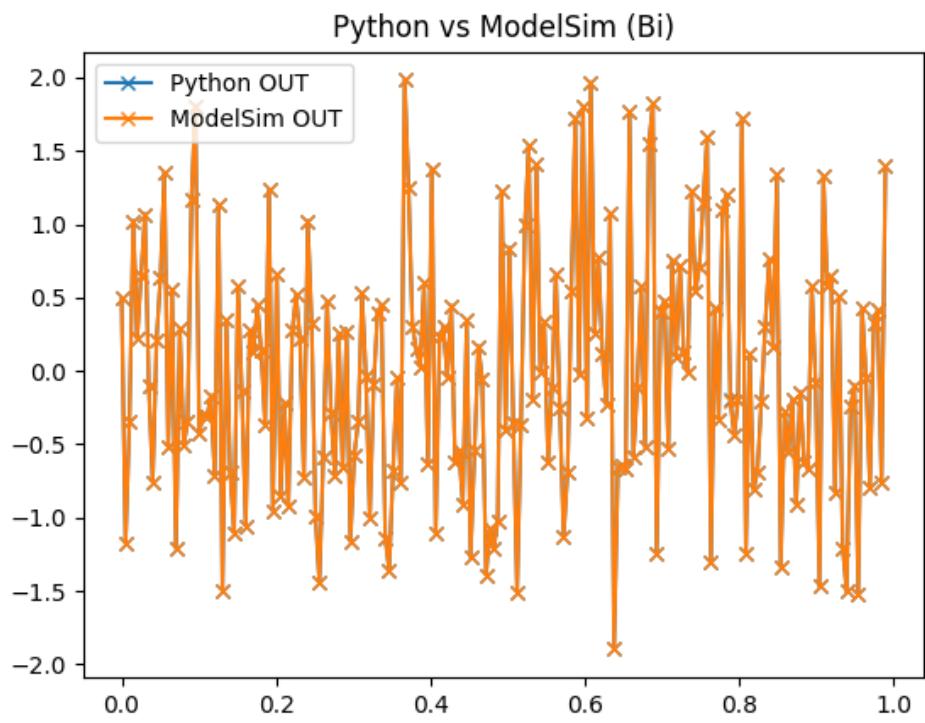


Figure 24: Simulazione Python e ModelSim per il calcolo di Bi_out

L’andamento degli errori, mostrato nelle figure 25, 26, 27 e 28.

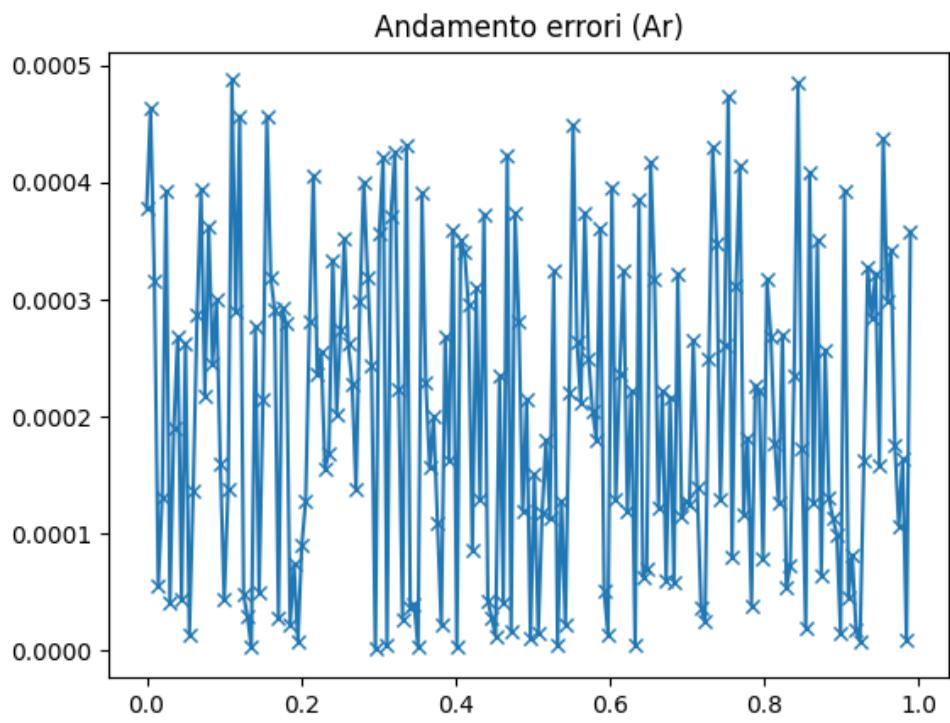


Figure 25: Andamento errore di A_r

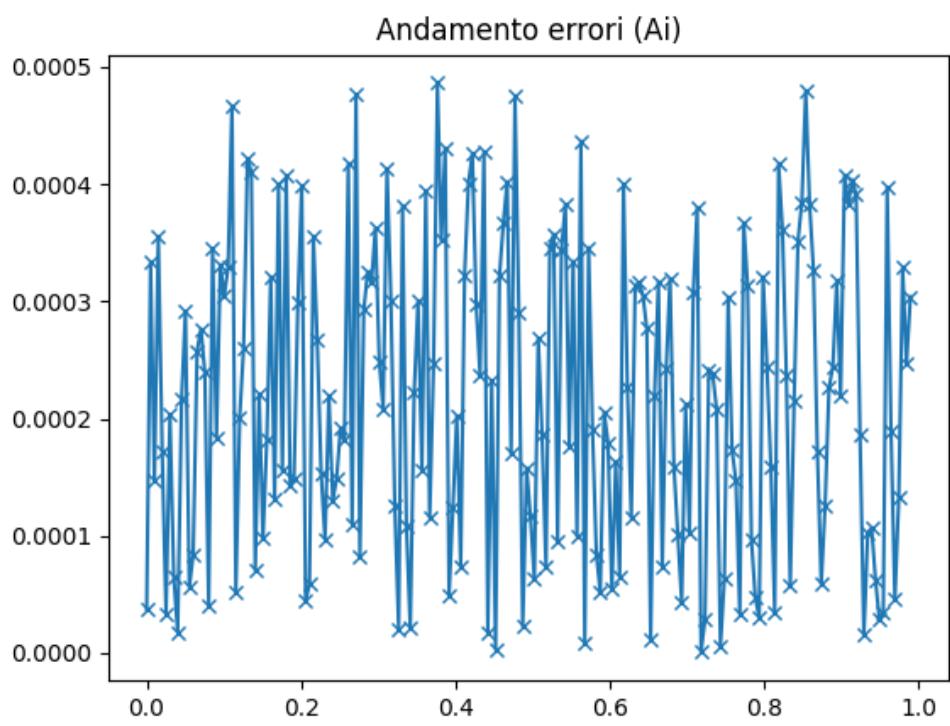


Figure 26: Andamento errore di A_i

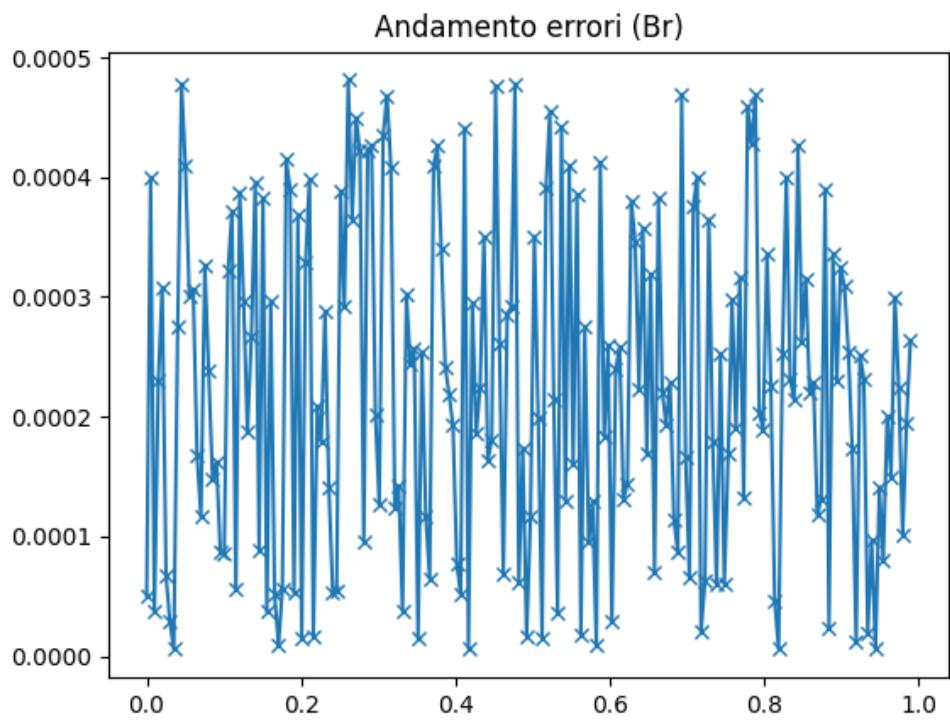


Figure 27: Andamento errore di Br

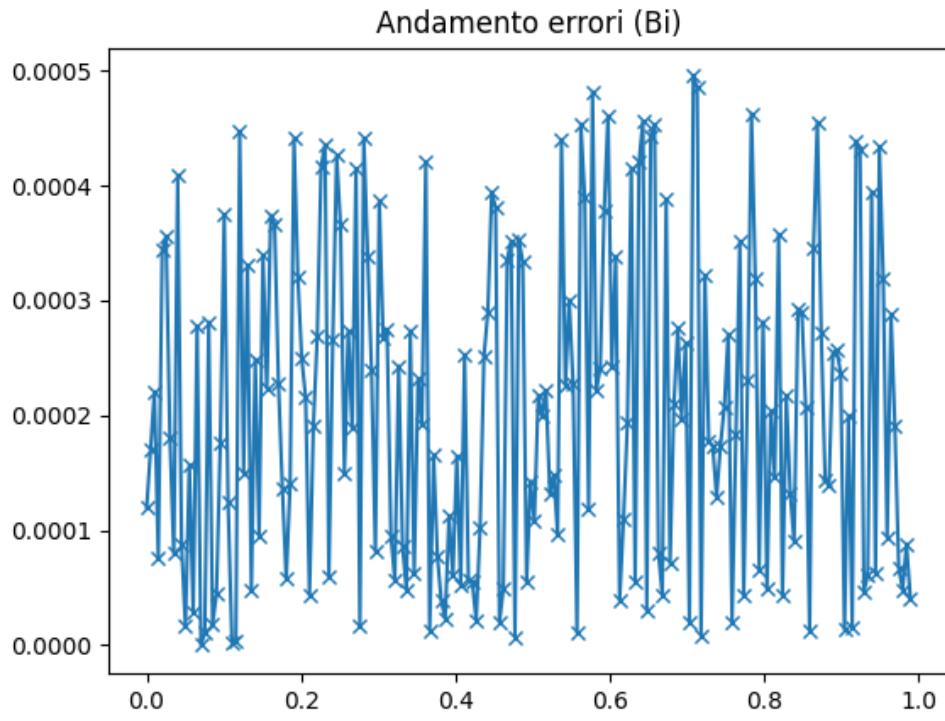


Figure 28: Andamento errore di B.i

7.2 FFT

Infine abbiamo testato ulteriormente la nostra Butterfly realizzando un'architettura in grado di calcolare una FFT a 16 campioni. Le Butterfly sono così collegate:

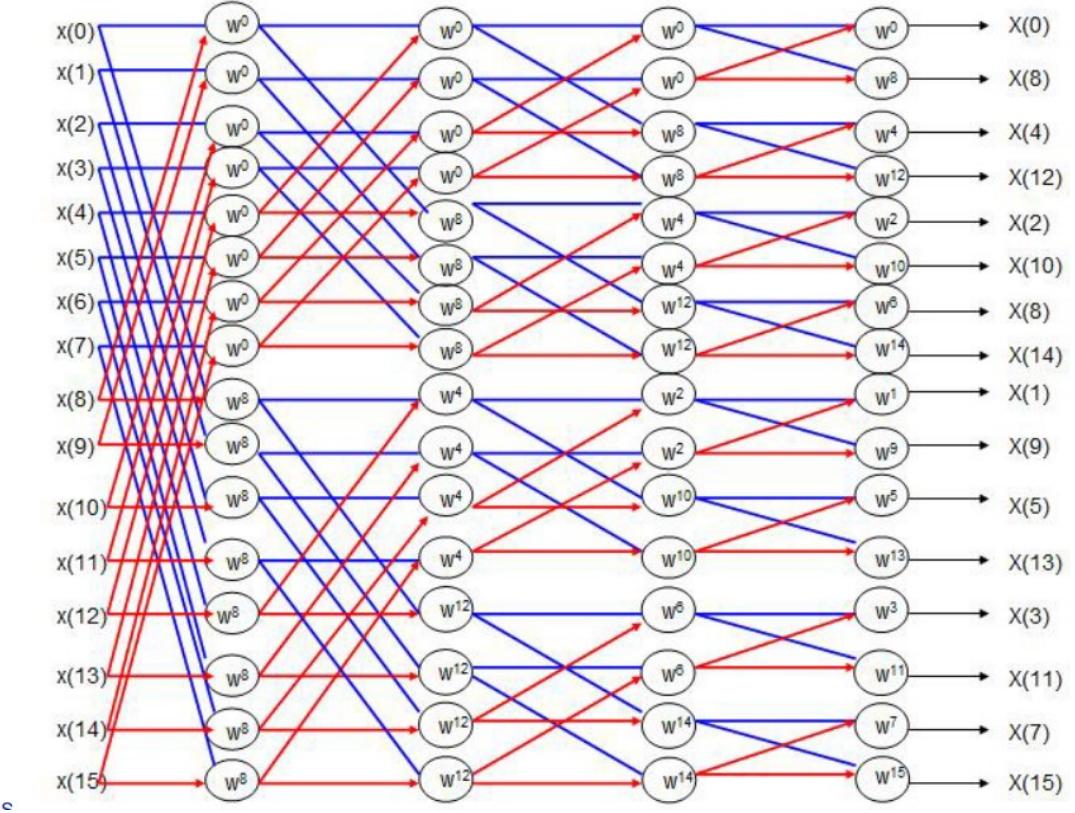


Figure 29: Struttura della FFT a 16 campioni

Per fare la simulazione in Python è stata programmata la funzione *butterfly* che ha gli stessi ingressi e le medesime uscite della Butterfly realizzata. La funzione *FFT* è stata ottenuta collegando ingressi e uscite di 32 funzioni *butterfly* sempre come in figura 29. Sono stati usati gli stessi 4 vettori di prova per le simulazioni del VHDL e in Python e come si può evincere dalle seguenti immagini, dove vengono paragonati i risultati di una simulazione in Python con quelli ottenuti con una simulazione ModelSim della nostra architettura, tutto funziona come dovrebbe.

	Input	Output VHDL	Output Phyton/16	Input	Output VHDL	Output Phyton/16	Input	Output VHDL	Output Phyton/16	Input	Output VHDL	Output Phyton/16
X0_R	0,5	0,0151367	0,015625	0,5	0,000976563	0,000976563	0,5	0,00146484	0,001953125	1	0,307617	0,03125
X0_I	0	0	0	0	0	0	0	0	0	1	0,307617	0,03125
X1_R	0,5	0	0	0	0,000976563	0,000976563	0,5	-0,000488281	6,72E-10	1	0	0
X1_I	0	0	0	0	0	0	0	-0,012539	-0,009819021	1	0	0
X2_R	0,5	0	0	0	0,000976563	0,000976563	0,5	0,00146484	0,001953125	1	0	0
X2_I	0	0	0	0	0	0	0	0	0	1	0	0
X3_R	0,5	0	0	0	0,000976563	0,000976563	0,5	-0,000488281	-1,20E-09	1	0	0
X3_I	0	0	0	0	0	0	0	-0,00292969	-0,002923058	1	0	0
X4_R	0,5	0	0	0	0,000976563	0,000976563	0,5	0,00146484	0,001953125	1	0	0
X4_I	0	0	0	0	0	0	0	0	0	1	0	0
X5_R	0,5	0	0	0	0,000976563	0,000976563	0,5	0	1,20E-09	1	0	0
X5_I	0	0	0	0	0	0	0	-0,00146484	-0,001305034	1	0	0
X6_R	0,5	0	0	0	0,000976563	0,000976563	0,5	0,00146484	0,001953125	1	0	0
X6_I	0	0	0	0	0	0	0	0	0	1	0	0
X7_R	0,5	0	0	0	0,000976563	0,000976563	0,5	0	-6,72E-10	1	0	0
X7_I	0	0	0	0	0	0	0	-0,000488281	-0,000388497	1	0	0
X8_R	0,5	0	0	0	0,000976563	0,000976563	0,5	0,00146484	0,001953125	1	0	0
X8_I	0	0	0	0	0	0	0	0	0	1	0	0
X9_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	0	-6,72E-10	1	0	0
X9_I	0	0	0	0	0	0	0	0	0	1	0	0
X10_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	0,00146484	0,001953125	1	0	0
X10_I	0	0	0	0	0	0	0	0	0	1	0	0
X11_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	0	-1,20E-09	1	0	0
X11_I	0	0	0	0	0	0	0	0,000976563	0,001305034	1	0	0
X12_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	0,00146484	0,001953125	1	0	0
X12_I	0	0	0	0	0	0	0	0	0	1	0	0
X13_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	-0,000488281	-1,20E-09	1	0	0
X13_I	0	0	0	0	0	0	0	0,00292969	0,002923058	1	0	0
X14_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	0,00146484	0,001953125	1	0	0
X14_I	0	0	0	0	0	0	0	0	0	1	0	0
X15_R	0,5	0	0	0	0,000976563	0,000976563	-0,5	-0,000488281	6,72E-10	1	0	0
X15_I	0	0	0	0	0	0	0	0,00976563	0,009819021	1	0	0

Figure 30: Simulazione della FFT

8 Appendix

8.1 pipe adder

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PIPE_ADDER is
port(
    CLK,RST,ADD_nDIFF: IN STD_LOGIC;
    A: IN std_logic_vector(32 downto 0);
    B: IN std_logic_vector(32 downto 0);
    SUM: OUT std_logic_vector(32 downto 0)
);
end PIPE_ADDER;

architecture behavioral of PIPE_ADDER is

signal a_signed, b_signed: signed(32 downto 0);
signal a_signed_resize, b_signed_resize: signed(32 downto 0);
signal sum_signed, sum_pipe: signed(32 downto 0);

begin
a_signed<=signed(A);
b_signed<=signed(B);
a_signed_resize<=resize(a_signed,33);
b_signed_resize<=resize(b_signed,33);
SUM<=std_logic_vector(sum_signed);

sum_output: process(CLK)
begin
    if(CLK'event and CLK='1') then
        if(RST='1') then
            sum_pipe<=(others=>'0');
            sum_signed<=(others=>'0');
        else
            if(ADD_nDIFF='1') then
                sum_pipe<=a_signed_resize+b_signed_resize;
                sum_signed<=sum_pipe;
            else
                sum_pipe<=a_signed_resize-b_signed_resize;
                sum_signed<=sum_pipe;
            end if;
        end if;
    end if;
end process;
end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.2 pipe mult

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PIPE_MULT is
port(
    CLK, RST, MULT_nSHIFT: IN STD_LOGIC;
    A,B: IN std_logic_vector(15 downto 0);
    MULT_OUT: OUT std_logic_vector(30 downto 0);
    SHIFT_OUT: OUT std_logic_vector(16 downto 0)
);
end PIPE_MULT;

architecture behavioral of PIPE_MULT is
signal a_signed, b_signed: signed(15 downto 0);
signal mult_out_signed, mult_pipe: signed(30 downto 0); -- (2n-1) bit
                                         perche' -1<A<1 e -1<B<1

begin
a_signed<=signed(A);
b_signed<=signed(B);
MULT_OUT<=std_logic_vector(mult_out_signed);
out_MULT: process(CLK)
begin

if(CLK'event and CLK='1') then
    if(RST='1') then
        mult_pipe<=(others=>'0');
        mult_out_signed<=(others=>'0');
        SHIFT_OUT<=(others=>'0');
    else
        if(MULT_nSHIFT='1') then
            mult_pipe<=resize(a_signed*b_signed,31);
            mult_out_signed<=mult_pipe;
-- il tool di sintesi in grado di ridistribuire autonomamente
-- la rete logica in due parti uguali
            else
                mult_out_signed<=mult_pipe;
                SHIFT_OUT<=A&'0';
            end if;
        end if;
    end if;
end process;

end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.3 ROM round

```
library ieee;
use ieee.std_logic_1164.all;

entity ROM_ROUND is
port(
    CLK: in std_logic;
    ADD: in std_logic_vector(3 downto 0);
    EN: in std_logic;
    Q: out std_logic_vector(2 downto 0)
);
end entity;

architecture behavioral of ROM_ROUND is

begin

process(CLK)
begin

if(rising_edge(CLK) and EN='1') then
    case ADD is
        when "0000" => Q <= "000";
        when "0001" => Q <= "001";
        when "0010" => Q <= "001";
        when "0011" => Q <= "010";
        when "0100" => Q <= "010";
        when "0101" => Q <= "011";
        when "0110" => Q <= "011";
        when "0111" => Q <= "100";
        when "1000" => Q <= "100";
        when "1001" => Q <= "100";
        when "1010" => Q <= "101";
        when "1011" => Q <= "101";
        when "1100" => Q <= "110";
        when "1101" => Q <= "110";
        when "1110" => Q <= "111";
        when "1111" => Q <= "111";
        when others =>
            end case;
    end if;
end process;
end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.4 round and scale

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity round_and_scale is
port
(
    clk      : in std_logic;
    block_in : in std_logic_vector(16 downto 0);
    enable:  in std_logic;
    SF_2h_11 : in std_logic;
    block_out : out std_logic_vector(15 downto 0)
);
end entity;

architecture behavioral of round_and_scale is

component ROM_ROUND is
port(
    CLK: in std_logic;
    ADD: in std_logic_vector(3 downto 0);
    EN: in std_logic;
    Q: out std_logic_vector(2 downto 0)
);
end component;

signal Q_Round  : std_logic_vector(2 downto 0);
signal Round_out: std_logic_vector(15 downto 0);

begin

Rom_prova:ROM_ROUND
port map(CLK=>clk,ADD=>block_in(3 downto 0),EN=>enable,Q=>Q_Round);

assign: process(clk)
begin
    if(rising_edge(clk)) then
        Round_out(15 downto 3)<=block_in(16 downto 4);
    end if;
end process;

Round_out(2 downto 0)<=Q_Round;

scale: process(Round_out,SF_2h_11)
begin
    if SF_2h_11='0' then
        block_out<=Round_out(15) & Round_out(15 downto 1);
    else
        block_out<=Round_out(15) & Round_out(15) & Round_out(15 downto 2);
    end if;
end process;
```

```
    end if;
end process;
end architecture;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.5 late status PLA

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Late_Status_PLA is
port(
    Status_START: IN STD_LOGIC;
    CC: IN STD_LOGIC;
    LSB: IN STD_LOGIC;
    CC_Validation: OUT STD_LOGIC;
    Out_LSP: OUT STD_LOGIC
);
end Late_Status_PLA;

architecture behavioral of Late_Status_PLA is
begin
process(Status_START,LSB,CC)
begin
    if CC='0' then
        Out_LSP<=LSB;
        CC_Validation<='0';
    else
        if (Status_START='1')then
            CC_Validation<='1';
            Out_LSP<='1';
        else
            CC_Validation<='0';
            Out_LSP<='0';
        end if;
    end if;
end process;
end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.6 uAddress Register

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uAddress_Register is
generic(
    N: integer := 8
);
port(
    CLK,RST: IN STD_LOGIC;
    Input: IN STD_LOGIC_vector(N downto 0);
    Output: OUT STD_LOGIC_vector(N downto 0)
);
end uAddress_Register;

architecture behavioral of uAddress_Register is
begin
process(CLK,RST)
begin
    if(RST='0') then
        Output<=(others=>'0');
    elsif(falling_edge(clk)) then
        Output<=Input;
    end if;
end process;
end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.7 uInstruction Register

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity uInstruction_Register is
generic(
    N: integer := 8
);
port(
    CLK,RST: IN STD_LOGIC;
    Input: IN STD_LOGIC_vector(N downto 0);
    Output: OUT STD_LOGIC_vector(N downto 0)
);
end uInstruction_Register;

architecture behavioral of uInstruction_Register is
begin
process(CLK,RST)
begin
    if(RST='0') then
        Output<="000010000000000000000001";
    elsif(rising_edge(clk)) then
        Output<=Input;
    end if;
end process;
end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.8 uROM

8.9 Control Unit

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity CU is
port(
    CLK: IN STD_LOGIC;
    START: IN STD_LOGIC;
    RST: IN STD_LOGIC;
    MULT_nSHIFT: out std_logic;
    ADD_nDIFF : out std_logic;
    ROUND_EN : out std_logic;
    sel1 : out std_logic;
    sel2 : out std_logic;
    sel3 : out std_logic;
    sel4 : out std_logic;
    sel5 : out std_logic;
    sel6 : out std_logic;
    sel7 : out std_logic;
    DONE:out std_logic;
    RST_B: out std_logic;
    LE_AR_OUT,LE_AI_OUT,LE_BR_OUT,LE_BI_OUT,LE_RF: out std_logic
);
end CU;

architecture behavioral of CU is

component Late_Status_PLA is
port(
    Status_START: IN STD_LOGIC;
    CC: IN STD_LOGIC;
    LSB: IN STD_LOGIC;
    CC_Validation: OUT STD_LOGIC;
    Out_LSP: OUT STD_LOGIC
);
end component;

component uAddress_Register is
generic(
    N: integer := 8
);
port(
    CLK,RST: IN STD_LOGIC;
    Input: IN STD_LOGIC_vector(N downto 0);
    Output: OUT STD_LOGIC_vector(N downto 0)
);
end component;

component uInstruction_Register is
generic(
    N: integer := 8
);
port(
```

```

CLK, RST: IN STD_LOGIC;
Input: IN STD_LOGIC_vector(N downto 0);
Output: OUT STD_LOGIC_vector(N downto 0)
);
end component;

component uROM is
port(
  ADD: in std_logic_vector(3 downto 0);
  Q: out std_logic_vector(45 downto 0)
);
end component;

-----
signal CC    : std_logic;
signal LSB   : std_logic;
signal CC_Validation : std_logic;
signal Out_LSP  : std_logic;

signal out_uAR:std_logic_vector(4 downto 0);
signal next_address:std_logic_vector(3 downto 0);

signal in_uIR:std_logic_vector(22 downto 0);

signal Q: std_logic_vector(45 downto 0);

signal out_MUX_CC :std_logic;
-----
begin

Status_PLA: Late_Status_PLA
port map(Status_START=>START, CC=>CC, LSB=>LSB, CC_Validation=>CC_Validation,
        Out_LSP=>Out_LSP);

ROM:uROM
port map(ADD=>out_uAR(4 downto 1),Q=>Q);

uAR: uAddress_Register
generic map(N=>4)
port map(CLK=>CLK,RST=>RST, Input(4 downto 1)=>next_address, Input(0)=>
        Out_LSP, Output=>out_uAR);

uIR: uInstruction_Register
generic map(N=>22)
port map(CLK=>CLK,RST=>RST, Input=>in_uIR, Output(22)=>CC, Output(21 downto 18
) =>next_address, Output(17)=>LSB,
        Output(16)=>LE_RF, Output(15)=>sel1,
        Output(14)=>sel2, Output(13)=>sel3,
        Output(12)=>sel4, Output(11)=>sel5, Output(10)=>sel6, Output(9)=>sel7,
        Output(8)=>MULT_nSHIFT, Output(7)=>
        ADD_nDIFF, Output(6)=>ROUND_EN,
        Output(5)=>LE_AR_OUT, Output(4)=>LE_AI_OUT, Output(3)=>LE_BR_OUT, Output(2)=>LE_BI_OUT, Output(1)=>DONE, Output(0)=>RST_B);

```

```

MUX_CC: process(out_uAR(0),Out_LSP,CC_Validation)
begin
  if(CC_Validation='0') then
    out_MUX_CC<=out_uAR(0);
  else
    out_MUX_CC<=Out_LSP;
  end if;
end process;

MUX_uROM: process(Q,out_MUX_CC)
begin
  if(out_MUX_CC='0') then
    in_uIR<=Q(45 downto 23);
  else
    in_uIR<=Q(22 downto 0);
  end if;
end process;

end behavioral;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.10 Butterfly

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.fixed_pkg.all;

entity Butterfly is
port
(
    clk      : in std_logic;
    Ar       : in std_logic_vector(15 downto 0);
    Ai       : in std_logic_vector(15 downto 0);
    Br       : in std_logic_vector(15 downto 0);
    Bi       : in std_logic_vector(15 downto 0);
    Wr       : in std_logic_vector(15 downto 0);
    Wi       : in std_logic_vector(15 downto 0);
    START    : in std_logic;
    RST      : in std_logic;
    SF_2h_1l: in std_logic;
    Ar_out   : buffer std_logic_vector(15 downto 0);
    Ai_out   : buffer std_logic_vector(15 downto 0);
    Br_out   : buffer std_logic_vector(15 downto 0);
    Bi_out   : buffer std_logic_vector(15 downto 0);
    DONE     : OUT std_logic
);
end entity;

architecture behavioral of Butterfly is

component PIPE_MULT is
port(
    CLK, RST, MULT_nSHIFT: IN STD_LOGIC;
    A,B: IN std_logic_vector(15 downto 0);
    MULT_OUT: OUT std_logic_vector(30 downto 0);
    SHIFT_OUT: OUT std_logic_vector(16 downto 0)
);
end component;

component PIPE_ADDER is
port(
    CLK,RST,ADD_nDIFF: IN STD_LOGIC;
    A: IN std_logic_vector(32 downto 0);
    B: IN std_logic_vector(32 downto 0);
    SUM: OUT std_logic_vector(32 downto 0)
);
end component;

component round_and_scale is
port
(
    clk      : in std_logic;
    block_in : in std_logic_vector(16 downto 0);
    enable   : in std_logic;
    SF_2h_1l : in std_logic;
```

```

    block_out : out std_logic_vector(15 downto 0)
  );
end component;

component CU is
port(
  CLK: IN STD_LOGIC;
  START: IN STD_LOGIC;
  RST: IN STD_LOGIC;
  MULT_nSHIFT: out std_logic;
  ADD_nDIFF : out std_logic;
  ROUND_EN : out std_logic;
  sel1 : out std_logic;
  sel2 : out std_logic;
  sel3 : out std_logic;
  sel4 : out std_logic;
  sel5 : out std_logic;
  sel6 : out std_logic;
  sel7 : out std_logic;
  DONE:out std_logic;
  RST_B: out std_logic;
  LE_AR_OUT,LE_AI_OUT,LE_BR_OUT,LE_BI_OUT,LE_RF: out std_logic
);
end component;

```

```

signal MULT_nSHIFT: std_logic;
signal MULT_OUT      : std_logic_vector(30 downto 0);
signal MULT_OUT_sfixed : sfixed(0 downto -30);
signal SHIFT_OUT : std_logic_vector(16 downto 0);

signal ADD_nDIFF : std_logic;
signal SUM         : std_logic_vector(32 downto 0);

signal round_in     : std_logic_vector(16 downto 0);
signal round_out    : std_logic_vector(15 downto 0);
signal ROUND_EN     : std_logic;
signal scale_select  : std_logic;--S_2H_1L

signal sel1 : std_logic;
signal sel2 : std_logic;
signal sel3 : std_logic;
signal sel4 : std_logic;
signal sel5 : std_logic;
signal sel6 : std_logic;
signal sel7 : std_logic;
signal outmux1: std_logic_vector(15 downto 0);
signal outmux1_sfixed: sfixed(0 downto -15);
signal outmux2: std_logic_vector(15 downto 0);
signal outmux3: std_logic_vector(15 downto 0);
signal outmux4: std_logic_vector(15 downto 0);
signal outmux5: std_logic_vector(32 downto 0);
signal outmux6: std_logic_vector(32 downto 0);

```

```

signal outmux7: std_logic_vector(32 downto 0);
signal inmux5: std_logic_vector(32 downto 0);
signal inmux5_sfixed: sfixed(2 downto -30);
signal inmux6: std_logic_vector(32 downto 0);
signal inmux6_sfixed: sfixed(2 downto -30);
signal inmux7: std_logic_vector(32 downto 0);
signal inmux7_sfixed: sfixed(2 downto -30);

signal output_shift_register_sfixed : sfixed(1 downto -15);
signal RST_B:std_logic;
signal A_r,A_i,B_i,B_r,W_r,W_i: std_logic_vector(15 downto 0);
signal LE_AR_OUT,LE_AI_OUT,LE_BR_OUT,LE_BI_OUT,LE_RF: std_logic;

-----
-----
```

begin

Moltiplicatore:PIPE_MULT
port map(CLK=>clk,RST=>RST_B,MULT_nSHIFT=>MULT_nSHIFT,A=>outmux3,B=>outmux4
,MULT_OUT=>MULT_OUT,SHIFT_OUT=>
SHIFT_OUT);

Sommatore:PIPE_ADDER
port map(CLK=>clk,RST=>RST_B,ADD_nDIFF=>ADD_nDIFF,A=>outmux6,B=>outmux7,SUM
=>SUM);

Rom_rounding: round_and_scale
port map(CLK=>clk,block_in=>round_in,enable=>ROUND_EN,SF_2h_1l=>
scale_select,block_out=>round_out);

MUX1: process(A_r,A_i,sel1)
begin
if(sel1='0') then
outmux1<=A_r;
else
outmux1<=A_i;
end if;
end process;

MUX2: process(B_r,B_i,sel2)
begin
if(sel2='0') then
outmux2<=B_r;
else
outmux2<=B_i;
end if;
end process;

MUX3: process(outmux2,outmux1,sel3)
begin
if(sel3='0') then
outmux3<=outmux2;
else

```

        outmux3<=outmux1;
    end if;
end process;

MUX4: process (W_r,W_i,sel4)
begin
if(sel4='0') then
    outmux4<=W_r;
else
    outmux4<=W_i;
end if;
end process;

output_shift_register_sfixed<=to_sfixed(SHIFT_OUT,
                                         output_shift_register_sfixed);
inmux5_sfixed<=resize(output_shift_register_sfixed,inmux5_sfixed);
inmux5<=to_std_logic_vector(inmux5_sfixed);

MUX5:process (sum,inmux5,sel5)
begin
if(sel5='0') then
    outmux5<=sum;
else
    outmux5<=inmux5;
end if;
end process;

outmux1_sfixed<=to_sfixed(outmux1,outmux1_sfixed);
inmux6_sfixed<=resize(outmux1_sfixed,inmux6_sfixed);
inmux6<=to_std_logic_vector(inmux6_sfixed);

MUX6:process (outmux5,inmux6,sel6)
begin
if(sel6='0') then
    outmux6<=outmux5;
else
    outmux6<=inmux6;
end if;
end process;

MULT_OUT_sfixed<=to_sfixed(MULT_OUT,MULT_OUT_sfixed);
inmux7_sfixed<=resize(MULT_OUT_sfixed,inmux7_sfixed);
inmux7<=to_std_logic_vector(inmux7_sfixed);

MUX7:process (inmux7,SUM,sel7)
begin
if(sel7='0') then
    outmux7<=inmux7;
else
    outmux7<=SUM;
end if;
end process;

round_in<=SUM(32 downto 16);

```

```

Control_Unit:CU
port map(CLK,START,RST,MULT_nSHIFT,ADD_nDIFF,ROUND_EN,sel1,sel2,sel3,sel4,
         sel5,sel6,sel7,DONE,RST_B,LE_AR_OUT,
         LE_AI_OUT,LE_BR_OUT,LE_BI_OUT,LE_RF);
-----
```

```

AR_IN_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      A_r<=(others=>'0');
    elsif(LE_RF='1') then
      A_r<=Ar;
    else
      A_r<=A_r;
    end if;
  end if;
end process;
```

```

AI_IN_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      A_i<=(others=>'0');
    elsif(LE_RF='1') then
      A_i<=Ai;
    else
      A_i<=A_i;
    end if;
  end if;
end process;
```

```

BR_IN_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      B_r<=(others=>'0');
    elsif(LE_RF='1') then
      B_r<=Br;
    else
      B_r<=B_r;
    end if;
  end if;
end process;
```

```

BI_IN_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      B_i<=(others=>'0');
    elsif(LE_RF='1') then

```

```

    B_i<=Bi;
  else
    B_i<=B_i;
  end if;
end if;
end process;

WR_IN_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      W_r<=(others=>'0');
    elsif(LE_RF='1') then
      W_r<=Wr;
    else
      W_r<=W_r;
    end if;
  end if;
end process;

WI_IN_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      W_i<=(others=>'0');
    elsif(LE_RF='1') then
      W_i<=Wi;
    else
      W_i<=W_i;
    end if;
  end if;
end process;

AR_OUT_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      AR_OUT<=(others=>'0');
    elsif(LE_AR_OUT='1') then
      AR_OUT<=round_out;
    else
      AR_OUT<=AR_OUT;
    end if;
  end if;
end process;

AI_OUT_REG: process(clk)
begin
  if(clk'event and clk='1') then
    if(RST_B='1') then
      AI_OUT<=(others=>'0');
    elsif(LE_AI_OUT='1') then
      AI_OUT<=round_out;
    else
      AI_OUT<=AI_OUT;
    end if;
  end if;
end process;

```

```

        end if;
    end if;
end process;

BR_OUT_REG: process(clk)
begin
    if(clk'event and clk='1') then
        if(RST_B='1') then
            BR_OUT<=(others=>'0');
        elsif(LE_BR_OUT='1') then
            BR_OUT<=round_out;
        else
            BR_OUT<=BR_OUT;
        end if;
    end if;
end process;

BI_OUT_REG: process(clk)
begin
    if(clk'event and clk='1') then
        if(RST_B='1') then
            BI_OUT<=(others=>'0');
        elsif(LE_BI_OUT='1') then
            BI_OUT<=round_out;
        else
            BI_OUT<=BI_OUT;
        end if;
    end if;
end process;

campiona_S_2h_11: process(clk)
begin
    if(rising_edge(clk)) then
        if(RST_B='1') then
            scale_select<='0';
        elsif(LE_RF='1') then
            scale_select<=SF_2h_11;
        end if;
    end if;
end process;

end architecture;

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----

```

8.11 FFT

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.fixed_pkg.all;

entity FFT is
port
(
    clk      : in std_logic;
    X0_R,X0_I,X1_R,X1_I,X2_R,X2_I,X3_R,X3_I,X4_R,X4_I,X5_R,X5_I,X6_R,X6_I,
    X7_R,X7_I,X8_R,X8_I: in
        std_logic_vector(15 downto 0);
    X9_R,X9_I,X10_R,X10_I,X11_R,X11_I,X12_R,X12_I,X13_R,X13_I,X14_R,X14_I,
    X15_R,X15_I: in std_logic_vector(15
        downto 0);

    START : in std_logic;
    RST   : in std_logic;
    X0_R_OUT,X0_I_OUT,X1_R_OUT,X1_I_OUT,X2_R_OUT,X2_I_OUT,X3_R_OUT,X3_I_OUT
        ,X4_R_OUT,X4_I_OUT,X5_R_OUT,X5_I_OUT,
        X6_R_OUT,X6_I_OUT,X7_R_OUT,X7_I_OUT,
        X8_R_OUT,X8_I_OUT: OUT
        std_logic_vector(15 downto 0);
    X9_R_OUT,X9_I_OUT,X10_R_OUT,X10_I_OUT,X11_R_OUT,X11_I_OUT,X12_R_OUT,
        X12_I_OUT,X13_R_OUT,X13_I_OUT,
        X14_R_OUT,X14_I_OUT,X15_R_OUT,
        X15_I_OUT: OUT std_logic_vector(15
        downto 0);

    DONE: OUT std_logic
);
end entity;

architecture behavioral of FFT is

component Butterfly is
port
(
    clk      : in std_logic;
    Ar      : in std_logic_vector(15 downto 0);
    Ai      : in std_logic_vector(15 downto 0);
    Br      : in std_logic_vector(15 downto 0);
    Bi      : in std_logic_vector(15 downto 0);
    Wr      : in std_logic_vector(15 downto 0);
    Wi      : in std_logic_vector(15 downto 0);
    START   : in std_logic;
    RST     : in std_logic;
    SF_2h_1l: in std_logic;
    Ar_out  : buffer std_logic_vector(15 downto 0);
    Ai_out  : buffer std_logic_vector(15 downto 0);
    Br_out  : buffer std_logic_vector(15 downto 0);
    Bi_out  : buffer std_logic_vector(15 downto 0);
    DONE: OUT std_logic
);
end component;
```

```

signal Wr0,Wr1,Wr2,Wr3,Wr4,Wr5,Wr6,Wr7 : std_logic_vector(15 downto 0);
signal Wi0,Wi1,Wi2,Wi3,Wi4,Wi5,Wi6,Wi7 : std_logic_vector(15 downto 0);
signal Wr0_sfixed,Wr1_sfixed,Wr2_sfixed,Wr3_sfixed,Wr4_sfixed,Wr5_sfixed,
       Wr6_sfixed,Wr7_sfixed : sfixed(0
       downto -15);
signal Wi0_sfixed,Wi1_sfixed,Wi2_sfixed,Wi3_sfixed,Wi4_sfixed,Wi5_sfixed,
       Wi6_sfixed,Wi7_sfixed : sfixed(0
       downto -15);
signal Y0_R,Y0_I,Y1_R,Y1_I,Y2_R,Y2_I,Y3_R,Y3_I,Y4_R,Y4_I,Y5_R,Y5_I,Y6_R,
       Y6_I,Y7_R,Y7_I,Y8_R,Y8_I:
       std_logic_vector(15 downto 0);
signal Y9_R,Y9_I,Y10_R,Y10_I,Y11_R,Y11_I,Y12_R,Y12_I,Y13_R,Y13_I,Y14_R,
       Y14_I,Y15_R,Y15_I: std_logic_vector(
       15 downto 0);
signal Z0_R,Z0_I,Z1_R,Z1_I,Z2_R,Z2_I,Z3_R,Z3_I,Z4_R,Z4_I,Z5_R,Z5_I,Z6_R,
       Z6_I,Z7_R,Z7_I,Z8_R,Z8_I:
       std_logic_vector(15 downto 0);
signal Z9_R,Z9_I,Z10_R,Z10_I,Z11_R,Z11_I,Z12_R,Z12_I,Z13_R,Z13_I,Z14_R,
       Z14_I,Z15_R,Z15_I: std_logic_vector(
       15 downto 0);
signal T0_R,T0_I,T1_R,T1_I,T2_R,T2_I,T3_R,T3_I,T4_R,T4_I,T5_R,T5_I,T6_R,
       T6_I,T7_R,T7_I,T8_R,T8_I:
       std_logic_vector(15 downto 0);
signal T9_R,T9_I,T10_R,T10_I,T11_R,T11_I,T12_R,T12_I,T13_R,T13_I,T14_R,
       T14_I,T15_R,T15_I: std_logic_vector(
       15 downto 0);
SIGNAL DONE_STADIO_1, DONE_STADIO_2,DONE_STADIO_3: STD_LOGIC;

begin

Wr0_sfixed<= (to_sfixed(1,Wr0_sfixed));
Wi0_sfixed<= (to_sfixed(0,Wi0_sfixed));
Wr0<=to_slv(Wr0_sfixed);
Wi0<=to_slv(Wi0_sfixed);
Wr1_sfixed<= (to_sfixed(0.923879,Wr1_sfixed));
Wi1_sfixed<= (to_sfixed(-0.382683,Wi1_sfixed));
Wr1<=to_slv(Wr1_sfixed);
Wi1<=to_slv(Wi1_sfixed);
Wr2_sfixed<= (to_sfixed(0.707107,Wr2_sfixed));
Wi2_sfixed<= (to_sfixed(-0.707107,Wi2_sfixed));
Wr2<=to_slv(Wr2_sfixed);
Wi2<=to_slv(Wi2_sfixed);
Wr3_sfixed<= (to_sfixed(0.382683,Wr3_sfixed));
Wi3_sfixed<= (to_sfixed(-0.923879,Wi3_sfixed));
Wr3<=to_slv(Wr3_sfixed);
Wi3<=to_slv(Wi3_sfixed);
Wr4_sfixed<= (to_sfixed(0,Wr4_sfixed));
Wi4_sfixed<= (to_sfixed(-1,Wi4_sfixed));
Wr4<=to_slv(Wr4_sfixed);
Wi4<=to_slv(Wi4_sfixed);
Wr5_sfixed<= (to_sfixed(-0.382683,Wr5_sfixed));
Wi5_sfixed<= (to_sfixed(-0.923879,Wi5_sfixed));
Wr5<=to_slv(Wr5_sfixed);
Wi5<=to_slv(Wi5_sfixed);

```

```

Wr6_sfixed<= (to_sfixed(-0.707107,Wr6_sfixed));
Wi6_sfixed<= (to_sfixed(-0.707107,Wi6_sfixed));
Wr6<=to_slv(Wr6_sfixed);
Wi6<=to_slv(Wi6_sfixed);
Wr7_sfixed<= (to_sfixed(-0.923879,Wr7_sfixed));
Wi7_sfixed<= (to_sfixed(-0.382683,Wi7_sfixed));
Wr7<=to_slv(Wr7_sfixed);
Wi7<=to_slv(Wi7_sfixed);

-----Stadio 1
Butterfly_1_0 :Butterfly
port map(clk=>clk,Ar=>X0_R,Ai=>X0_I,Br=>X8_R,Bi=>X8_I,Wr=>Wr0,Wi=>WI0,START
          =>START,RST=>RST,SF_2h_11=>'1',Ar_out
          =>Y0_R,Ai_out=>Y0_I,Br_out=>Y8_R,
          Bi_out=>Y8_I,DONE=>DONE_STADIO_1);
Butterfly_1_1 :Butterfly
port map(clk=>clk,Ar=>X1_R,Ai=>X1_I,Br=>X9_R,Bi=>X9_I,Wr=>Wr0,Wi=>WI0,START
          =>START,RST=>RST,SF_2h_11=>'1',Ar_out
          =>Y1_R,Ai_out=>Y1_I,Br_out=>Y9_R,
          Bi_out=>Y9_I,DONE=>DONE_STADIO_1);
Butterfly_1_2 :Butterfly
port map(clk=>clk,Ar=>X2_R,Ai=>X2_I,Br=>X10_R,Bi=>X10_I,Wr=>Wr0,Wi=>WI0,
          START=>START,RST=>RST,SF_2h_11=>'1',
          Ar_out=>Y2_R,Ai_out=>Y2_I,Br_out=>
          Y10_R,Bi_out=>Y10_I,DONE=>
          DONE_STADIO_1);
Butterfly_1_3 :Butterfly
port map(clk=>clk,Ar=>X3_R,Ai=>X3_I,Br=>X11_R,Bi=>X11_I,Wr=>Wr0,Wi=>WI0,
          START=>START,RST=>RST,SF_2h_11=>'1',
          Ar_out=>Y3_R,Ai_out=>Y3_I,Br_out=>
          Y11_R,Bi_out=>Y11_I,DONE=>
          DONE_STADIO_1);
Butterfly_1_4 :Butterfly
port map(clk=>clk,Ar=>X4_R,Ai=>X4_I,Br=>X12_R,Bi=>X12_I,Wr=>Wr0,Wi=>WI0,
          START=>START,RST=>RST,SF_2h_11=>'1',
          Ar_out=>Y4_R,Ai_out=>Y4_I,Br_out=>
          Y12_R,Bi_out=>Y12_I,DONE=>
          DONE_STADIO_1);
Butterfly_1_5 :Butterfly
port map(clk=>clk,Ar=>X5_R,Ai=>X5_I,Br=>X13_R,Bi=>X13_I,Wr=>Wr0,Wi=>WI0,
          START=>START,RST=>RST,SF_2h_11=>'1',
          Ar_out=>Y5_R,Ai_out=>Y5_I,Br_out=>
          Y13_R,Bi_out=>Y13_I,DONE=>
          DONE_STADIO_1);
Butterfly_1_6 :Butterfly
port map(clk=>clk,Ar=>X6_R,Ai=>X6_I,Br=>X14_R,Bi=>X14_I,Wr=>Wr0,Wi=>WI0,
          START=>START,RST=>RST,SF_2h_11=>'1',
          Ar_out=>Y6_R,Ai_out=>Y6_I,Br_out=>
          Y14_R,Bi_out=>Y14_I,DONE=>
          DONE_STADIO_1);
Butterfly_1_7 :Butterfly
port map(clk=>clk,Ar=>X7_R,Ai=>X7_I,Br=>X15_R,Bi=>X15_I,Wr=>Wr0,Wi=>WI0,
          START=>START,RST=>RST,SF_2h_11=>'1',
          Ar_out=>Y7_R,Ai_out=>Y7_I,Br_out=>
          Y15_R,Bi_out=>Y15_I,DONE=>
          
```

```

        DONE_STADIO_1);

-----Stadio 2
Butterfly_2_0 : Butterfly
port map (clk=>clk, Ar=>Y0_R, Ai=>Y0_I, Br=>Y4_R, Bi=>Y4_I, Wr=>Wr0, Wi=>WI0, START
          =>DONE_STADIO_1, RST=>RST, SF_2h_11=>'0
          ', Ar_out=>Z0_R, Ai_out=>Z0_I, Br_out=>
          Z4_R, Bi_out=>Z4_I, DONE=>DONE_STADIO_2
        );

Butterfly_2_1 : Butterfly
port map (clk=>clk, Ar=>Y1_R, Ai=>Y1_I, Br=>Y5_R, Bi=>Y5_I, Wr=>Wr0, Wi=>WI0, START
          =>DONE_STADIO_1, RST=>RST, SF_2h_11=>'0
          ', Ar_out=>Z1_R, Ai_out=>Z1_I, Br_out=>
          Z5_R, Bi_out=>Z5_I, DONE=>DONE_STADIO_2
        );

Butterfly_2_2 : Butterfly
port map (clk=>clk, Ar=>Y2_R, Ai=>Y2_I, Br=>Y6_R, Bi=>Y6_I, Wr=>Wr0, Wi=>WI0, START
          =>DONE_STADIO_1, RST=>RST, SF_2h_11=>'0
          ', Ar_out=>Z2_R, Ai_out=>Z2_I, Br_out=>
          Z6_R, Bi_out=>Z6_I, DONE=>DONE_STADIO_2
        );

Butterfly_2_3 : Butterfly
port map (clk=>clk, Ar=>Y3_R, Ai=>Y3_I, Br=>Y7_R, Bi=>Y7_I, Wr=>Wr0, Wi=>WI0, START
          =>DONE_STADIO_1, RST=>RST, SF_2h_11=>'0
          ', Ar_out=>Z3_R, Ai_out=>Z3_I, Br_out=>
          Z7_R, Bi_out=>Z7_I, DONE=>DONE_STADIO_2
        );

Butterfly_2_4 : Butterfly
port map (clk=>clk, Ar=>Y8_R, Ai=>Y8_I, Br=>Y12_R, Bi=>Y12_I, Wr=>Wr4, Wi=>WI4,
          START=>DONE_STADIO_1, RST=>RST,
          SF_2h_11=>'0', Ar_out=>Z8_R, Ai_out=>
          Z8_I, Br_out=>Z12_R, Bi_out=>Z12_I, DONE
          =>DONE_STADIO_2);

Butterfly_2_5 : Butterfly
port map (clk=>clk, Ar=>Y9_R, Ai=>Y9_I, Br=>Y13_R, Bi=>Y13_I, Wr=>Wr4, Wi=>WI4,
          START=>DONE_STADIO_1, RST=>RST,
          SF_2h_11=>'0', Ar_out=>Z9_R, Ai_out=>
          Z9_I, Br_out=>Z13_R, Bi_out=>Z13_I, DONE
          =>DONE_STADIO_2);

Butterfly_2_6 : Butterfly
port map (clk=>clk, Ar=>Y10_R, Ai=>Y10_I, Br=>Y14_R, Bi=>Y14_I, Wr=>Wr4, Wi=>WI4,
          START=>DONE_STADIO_1, RST=>RST,
          SF_2h_11=>'0', Ar_out=>Z10_R, Ai_out=>
          Z10_I, Br_out=>Z14_R, Bi_out=>Z14_I,
          DONE=>DONE_STADIO_2);

Butterfly_2_7 : Butterfly
port map (clk=>clk, Ar=>Y11_R, Ai=>Y11_I, Br=>Y15_R, Bi=>Y15_I, Wr=>Wr4, Wi=>WI4,
          START=>DONE_STADIO_1, RST=>RST,
          SF_2h_11=>'0', Ar_out=>Z11_R, Ai_out=>
          Z11_I, Br_out=>Z15_R, Bi_out=>Z15_I,
          DONE=>DONE_STADIO_2);

-----Stadio 3
Butterfly_3_0: Butterfly
port map (clk=>clk, Ar=>Z0_R, Ai=>Z0_I, Br=>Z2_R, Bi=>Z2_I, Wr=>Wr0, Wi=>WI0, START
          =>DONE_STADIO_2, RST=>RST, SF_2h_11=>'0
          ', Ar_out=>T0_R, Ai_out=>T0_I, Br_out=>

```

```

        T2_R, Bi_out=>T2_I, DONE=>DONE_STADIO_3
    );
Butterfly_3_1: Butterfly
port map(clk=>clk, Ar=>Z1_R, Ai=>Z1_I, Br=>Z3_R, Bi=>Z3_I, Wr=>Wr0, Wi=>Wi0, START
=>DONE_STADIO_2, RST=>RST, SF_2h_11=>'0
', Ar_out=>T1_R, Ai_out=>T1_I, Br_out=>
T3_R, Bi_out=>T3_I, DONE=>DONE_STADIO_3
);
Butterfly_3_2: Butterfly
port map(clk=>clk, Ar=>Z4_R, Ai=>Z4_I, Br=>Z6_R, Bi=>Z6_I, Wr=>Wr4, Wi=>Wi4, START
=>DONE_STADIO_2, RST=>RST, SF_2h_11=>'0
', Ar_out=>T4_R, Ai_out=>T4_I, Br_out=>
T6_R, Bi_out=>T6_I, DONE=>DONE_STADIO_3
);
Butterfly_3_3: Butterfly
port map(clk=>clk, Ar=>Z5_R, Ai=>Z5_I, Br=>Z7_R, Bi=>Z7_I, Wr=>Wr4, Wi=>Wi4, START
=>DONE_STADIO_2, RST=>RST, SF_2h_11=>'0
', Ar_out=>T5_R, Ai_out=>T5_I, Br_out=>
T7_R, Bi_out=>T7_I, DONE=>DONE_STADIO_3
);
Butterfly_3_4: Butterfly
port map(clk=>clk, Ar=>Z8_R, Ai=>Z8_I, Br=>Z10_R, Bi=>Z10_I, Wr=>Wr2, Wi=>Wi2,
START=>DONE_STADIO_2, RST=>RST,
SF_2h_11=>'0', Ar_out=>T8_R, Ai_out=>
T8_I, Br_out=>T10_R, Bi_out=>T10_I, DONE
=>DONE_STADIO_3);
Butterfly_3_5: Butterfly
port map(clk=>clk, Ar=>Z9_R, Ai=>Z9_I, Br=>Z11_R, Bi=>Z11_I, Wr=>Wr2, Wi=>Wi2,
START=>DONE_STADIO_2, RST=>RST,
SF_2h_11=>'0', Ar_out=>T9_R, Ai_out=>
T9_I, Br_out=>T11_R, Bi_out=>T11_I, DONE
=>DONE_STADIO_3);
Butterfly_3_6: Butterfly
port map(clk=>clk, Ar=>Z12_R, Ai=>Z12_I, Br=>Z14_R, Bi=>Z14_I, Wr=>Wr6, Wi=>Wi6,
START=>DONE_STADIO_2, RST=>RST,
SF_2h_11=>'0', Ar_out=>T12_R, Ai_out=>
T12_I, Br_out=>T14_R, Bi_out=>T14_I,
DONE=>DONE_STADIO_3);
Butterfly_3_7: Butterfly
port map(clk=>clk, Ar=>Z13_R, Ai=>Z13_I, Br=>Z15_R, Bi=>Z15_I, Wr=>Wr6, Wi=>Wi6,
START=>DONE_STADIO_2, RST=>RST,
SF_2h_11=>'0', Ar_out=>T13_R, Ai_out=>
T13_I, Br_out=>T15_R, Bi_out=>T15_I,
DONE=>DONE_STADIO_3);
-----Stadio 4
Butterfly_4_0: Butterfly
port map(clk=>clk, Ar=>T0_R, Ai=>T0_I, Br=>T1_R, Bi=>T1_I, Wr=>Wr0, Wi=>Wi0, START
=>DONE_STADIO_3, RST=>RST, SF_2h_11=>'0
', Ar_out=>X0_R_OUT, Ai_out=>X0_I_OUT,
Br_out=>X8_R_OUT, Bi_out=>X8_I_OUT,
DONE=>DONE);
Butterfly_4_1: Butterfly
port map(clk=>clk, Ar=>T2_R, Ai=>T2_I, Br=>T3_R, Bi=>T3_I, Wr=>Wr4, Wi=>Wi4, START
=>DONE_STADIO_3, RST=>RST, SF_2h_11=>'0
', Ar_out=>X4_R_OUT, Ai_out=>X4_I_OUT,

```

```

Br_out=>X12_R_OUT,Bi_out=>X12_I_OUT,
DONE=>DONE);

Butterfly_4_2: Butterfly
port map(clk=>clk,Ar=>T4_R,Ai=>T4_I,Br=>T5_R,Bi=>T5_I,Wr=>Wr2,Wi=>WI2,START
=>DONE_STADIO_3,RST=>RST,SF_2h_11=>'0
',Ar_out=>X2_R_OUT,Ai_out=>X2_I_OUT,
Br_out=>X10_R_OUT,Bi_out=>X10_I_OUT,
DONE=>DONE);

Butterfly_4_3: Butterfly
port map(clk=>clk,Ar=>T6_R,Ai=>T6_I,Br=>T7_R,Bi=>T7_I,Wr=>Wr6,Wi=>WI6,START
=>DONE_STADIO_3,RST=>RST,SF_2h_11=>'0
',Ar_out=>X6_R_OUT,Ai_out=>X6_I_OUT,
Br_out=>X14_R_OUT,Bi_out=>X14_I_OUT,
DONE=>DONE);

Butterfly_4_4: Butterfly
port map(clk=>clk,Ar=>T8_R,Ai=>T8_I,Br=>T9_R,Bi=>T9_I,Wr=>Wr1,Wi=>WI1,START
=>DONE_STADIO_3,RST=>RST,SF_2h_11=>'0
',Ar_out=>X1_R_OUT,Ai_out=>X1_I_OUT,
Br_out=>X9_R_OUT,Bi_out=>X9_I_OUT,
DONE=>DONE);

Butterfly_4_5: Butterfly
port map(clk=>clk,Ar=>T10_R,Ai=>T10_I,Br=>T11_R,Bi=>T11_I,Wr=>Wr5,Wi=>WI5,
START=>DONE_STADIO_3,RST=>RST,
SF_2h_11=>'0',Ar_out=>X5_R_OUT,Ai_out
=>X5_I_OUT,Br_out=>X13_R_OUT,Bi_out=>
X13_I_OUT,DONE=>DONE);

Butterfly_4_6: Butterfly
port map(clk=>clk,Ar=>T12_R,Ai=>T12_I,Br=>T13_R,Bi=>T13_I,Wr=>Wr3,Wi=>WI3,
START=>DONE_STADIO_3,RST=>RST,
SF_2h_11=>'0',Ar_out=>X3_R_OUT,Ai_out
=>X3_I_OUT,Br_out=>X11_R_OUT,Bi_out=>
X11_I_OUT,DONE=>DONE);

Butterfly_4_7: Butterfly
port map(clk=>clk,Ar=>T14_R,Ai=>T14_I,Br=>T15_R,Bi=>T15_I,Wr=>Wr7,Wi=>WI7,
START=>DONE_STADIO_3,RST=>RST,
SF_2h_11=>'0',Ar_out=>X7_R_OUT,Ai_out
=>X7_I_OUT,Br_out=>X15_R_OUT,Bi_out=>
X15_I_OUT,DONE=>DONE);

end architecture;

-----
-- GRUPPO 9: Palma Paolo 306046
-- Porcaro Mario 315888
-- Rinieri Filippo 317894
-----
```

8.12 TestButterfly.py

```
import numpy as np
from numpy import random
import matplotlib.pyplot as plt
from math import sqrt
import os
import subprocess

a_r = ((random.rand(200) - 0.5)*2)
a_i = ((random.rand(200) - 0.5)*2)
b_r = ((random.rand(200) - 0.5)*2)
b_i = ((random.rand(200) - 0.5)*2)
w_r = ((random.rand(200) - 0.5)*2)

w_i = []
i=0

#VETTORI PER PLOT
vecAR_py = []
vecAI_py = []
vecBR_py = []
vecBI_py = []
vecAR_sim = []
vecAI_sim = []
vecBR_sim = []
vecBI_sim = []
vecErr_AR = []
vecErr_AI = []
vecErr_BR = []
vecErr_BI = []

file_input_A_R = "inputs_A_R.txt"
file_input_A_I = "inputs_A_I.txt"
file_input_B_R = "inputs_B_R.txt"
file_input_B_I = "inputs_B_I.txt"
file_input_W_R = "inputs_W_R.txt"
file_input_W_I = "inputs_W_I.txt"

file_output_A_R_py = "output_A_R_py.txt"
file_output_A_I_py = "output_A_I_py.txt"
file_output_B_R_py = "output_B_R_py.txt"
file_output_B_I_py = "output_B_I_py.txt"

with open(file_input_A_R, 'w') as p1:
    with open(file_input_A_I, 'w') as p2:
        with open(file_input_B_R, 'w') as p3:
            with open(file_input_B_I, 'w') as p4:
                with open(file_input_W_R, 'w') as p5:
                    with open(file_input_W_I, 'w') as p6:
                        with open(file_output_A_R_py, 'w') as p7:
                            with open(file_output_A_I_py, 'w') as p8:
                                with open(file_output_B_R_py, 'w') as p9:
                                    with open(file_output_B_I_py, 'w') as p10:
```



```

file_log_B_I = "log_B_I.txt"

#CONTATORI ERRORI
i1=0
i2=0
i3=0
i4=0

with open(file_output_A_R_py, 'r') as p1:
    with open(file_output_A_I_py, 'r') as p2:
        with open(file_output_B_R_py, 'r') as p3:
            with open(file_output_B_I_py, 'r') as p4:
                with open(file_output_A_R_sim, 'r') as p5:
                    with open(file_output_A_I_sim, 'r') as p6:
                        with open(file_output_B_R_sim, 'r') as p7:
                            with open(file_output_B_I_sim, 'r') as p8:
                                with open(file_log_A_R, 'w') as p9:
                                    with open(file_log_A_I, 'w') as p10:
                                        with open(file_log_B_R, 'w') as p11:
                                            with open(file_log_B_I, 'w') as p12:
                                                for AR_py, AI_py, BR_py, BI_py, AR_sim, AI_sim, BR_sim, BI_sim in zip(
                                                    p1, p2, p3, p4, p5, p6, p7, p8):
                                                    # RIEMPIO VETTORI PER PLOT Py
                                                    vecAR_sim.append(float(AR_sim)*4)
                                                    vecAI_sim.append(float(AI_sim)*4)
                                                    vecBR_sim.append(float(BR_sim)*4)
                                                    vecBI_sim.append(float(BI_sim)*4)
                                                    vecErr_AR.append(abs(float(AR_py)-float(AR_sim)*4))
                                                    vecErr_AI.append(abs(float(AI_py)-float(AI_sim)*4))
                                                    vecErr_BR.append(abs(float(BR_py)-float(BR_sim)*4))
                                                    vecErr_BI.append(abs(float(BI_py)-float(BI_sim)*4))
                                                    if abs(float(AR_py)-float(AR_sim)*4) > pow(2,-11) :
                                                        print(float(AR_py), '\t', float(AR_sim)*4, '\t', "--ERRORE--", '\n', file=p9)
                                                        i1+=1
                                                    else:
                                                        print(float(AR_py), '\t', float(AR_sim)*4, '\t', "--OK--", '\n', file=p9)
                                                    if abs(float(AI_py)-float(AI_sim)*4) > pow(2,-11) :
                                                        print(float(AI_py), '\t', float(AI_sim)*4, '\t', "--ERRORE--", '\n', file=p10)
                                                        i2+=1
                                                    else:
                                                        print(float(AI_py), '\t', float(AI_sim)*4, '\t', "--OK--", '\n', file=p10)
                                                    if abs(float(BR_py)-float(BR_sim)*4) > pow(2,-11) :
                                                        print(float(BR_py), '\t', float(BR_sim)*4, '\t', "--ERRORE--", '\n', file=p11)
                                                        i3+=1
                                                    else:
                                                        print(float(BR_py), '\t', float(BR_sim)*4, '\t', "--OK--", '\n', file=p11)
                                                    if abs(float(BI_py)-float(BI_sim)*4) > pow(2,-11) :
                                                        print(float(BI_py), '\t', float(BI_sim)*4, '\t', "--ERRORE--", '\n', file=p12)

```

```

    i4+=1
else:
    print(float(BI_py), '\t', float(BI_sim)*4, '\t', "--OK--", '\n',
          , file=p12)
p12.close()
p11.close()
p10.close()
p9.close()
p8.close()
p7.close()
p6.close()
p5.close()
p4.close()
p3.close()
p2.close()
p1.close()

print("Percentuale output Ar con errore > 2^-11 = %.3f %%" % (i1*100/198))
print("Percentuale output Ai con errore > 2^-11 = %.3f %%" % (i2*100/198))
print("Percentuale output Br con errore > 2^-11 = %.3f %%" % (i3*100/198))
print("Percentuale output Bi con errore > 2^-11 = %.3f %%" % (i4*100/198))

#TIME BASE PER PLOT

time_base = np.linspace(0,1,200)

#PLOT AR

plt.figure(1)
plt.plot(time_base[:-2],vecAR_py[:-2],"-x",label="Python OUT")
plt.plot(time_base[:-2],vecAR_sim,"-x",label="ModelSim OUT")
plt.title("Python vs ModelSim (Ar)")
plt.legend()

#PLOT AI

plt.figure(2)
plt.plot(time_base[:-2],vecAI_py[:-2],"-x",label="Python OUT")
plt.plot(time_base[:-2],vecAI_sim,"-x",label="ModelSim OUT")
plt.title("Python vs ModelSim (Ai)")
plt.legend()

#PLOT BR

plt.figure(3)
plt.plot(time_base[:-2],vecBR_py[:-2],"-x",label="Python OUT")
plt.plot(time_base[:-2],vecBR_sim,"-x",label="ModelSim OUT")
plt.title("Python vs ModelSim (Br)")
plt.legend()

#PLOT BI

plt.figure(4)
plt.plot(time_base[:-2],vecBI_py[:-2],"-x",label="Python OUT")
plt.plot(time_base[:-2],vecBI_sim,"-x",label="ModelSim OUT")

```

```

plt.title("Python vs ModelSim (Bi)")
plt.legend()

#PLOT ERROR AR

plt.figure(5)
plt.plot(time_base[:-2],vecErr_AR, "-x")
plt.title("Andamento errori (Ar)")

#PLOT ERROR AI

plt.figure(6)
plt.plot(time_base[:-2],vecErr_AI, "-x")
plt.title("Andamento errori (Ai)")

#PLOT ERROR BR

plt.figure(7)
plt.plot(time_base[:-2],vecErr_BR, "-x")
plt.title("Andamento errori (Br)")

#PLOT ERROR BI

plt.figure(8)
plt.plot(time_base[:-2],vecErr_BI, "-x")
plt.title("Andamento errori (Bi)")

plt.show()

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----

```

8.13 compile.do

```
vcom PIPE_MULT.vhd
vcom ROM_ROUND.vhd
vcom round_and_scale.vhd
vcom uAddress_Register.vhd
vcom uInstruction_Register.vhd
vcom uROM.vhd
vcom PIPE_ADDER.vhd
vcom CU.vhd
vcom Butterfly.vhd
vcom Butterfly_tb.vhd

vsim -c work.Butterfly_tb

run 0ns
run 30us

#write list counter.lst
quit -f

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```

8.14 FFT TEST

```
W_0_R = 1
W_0_I = 0
W_1_R = 0.923879
W_1_I = -0.382683
W_2_R = 0.707107
W_2_I = -0.707107
W_3_R = 0.382683
W_3_I = -0.923879
W_4_R = 0
W_4_I = -1
W_5_R = -0.382683
W_5_I = -0.923879
W_6_R = -0.707107
W_6_I = -0.707107
W_7_R = -0.923879
W_7_I = -0.382683

#####
X_0_R = 1
X_0_I = 1
X_1_R = 1
X_1_I = 1
X_2_R = 1
X_2_I = 1
X_3_R = 1
X_3_I = 1
X_4_R = 1
X_4_I = 1
X_5_R = 1
X_5_I = 1
X_6_R = 1
X_6_I = 1
X_7_R = 1
X_7_I = 1
X_8_R = 1
X_8_I = 1
X_9_R = 1
X_9_I = 1
X_10_R = 1
X_10_I = 1
X_11_R = 1
X_11_I = 1
X_12_R = 1
X_12_I = 1
X_13_R = 1
X_13_I = 1
X_14_R = 1
X_14_I = 1
X_15_R = 1
X_15_I = 1

#####
```

```

def butterfly(a_r, a_i, b_r, b_i, w_r, w_i, Scale):
    out_AR = (a_r + (b_r * w_r) - (b_i * w_i))
    out_AI = (a_i + (b_r * w_i) + (b_i * w_r))
    out_BR = (-out_AR + 2 * a_r)
    out_BI = (2 * a_i - out_AI)
    if Scale == 1:
        out_AR_S = out_AR / 4
        out_AI_S = out_AI / 4
        out_BR_S = out_BR / 4
        out_BI_S = out_BI / 4
    else:
        out_AR_S = out_AR / 2
        out_AI_S = out_AI / 2
        out_BR_S = out_BR / 2
        out_BI_S = out_BI / 2
    return out_AR_S, out_AI_S, out_BR_S, out_BI_S

# Step 1
Y0_R, Y0_I, Y8_R, Y8_I = butterfly(X_0_R, X_0_I, X_8_R, X_8_I, W_0_R, W_0_I
                                      , 1)
Y1_R, Y1_I, Y9_R, Y9_I = butterfly(X_1_R, X_1_I, X_9_R, X_9_I, W_0_R, W_0_I
                                      , 1)
Y2_R, Y2_I, Y10_R, Y10_I = butterfly(X_2_R, X_2_I, X_10_R, X_10_I, W_0_R,
                                       W_0_I, 1)
Y3_R, Y3_I, Y11_R, Y11_I = butterfly(X_3_R, X_3_I, X_11_R, X_11_I, W_0_R,
                                       W_0_I, 1)
Y4_R, Y4_I, Y12_R, Y12_I = butterfly(X_4_R, X_4_I, X_12_R, X_12_I, W_0_R,
                                       W_0_I, 1)
Y5_R, Y5_I, Y13_R, Y13_I = butterfly(X_5_R, X_5_I, X_13_R, X_13_I, W_0_R,
                                       W_0_I, 1)
Y6_R, Y6_I, Y14_R, Y14_I = butterfly(X_6_R, X_6_I, X_14_R, X_14_I, W_0_R,
                                       W_0_I, 1)
Y7_R, Y7_I, Y15_R, Y15_I = butterfly(X_7_R, X_7_I, X_15_R, X_15_I, W_0_R,
                                       W_0_I, 1)

# Step 2
Z0_R, Z0_I, Z4_R, Z4_I = butterfly(Y0_R, Y0_I, Y4_R, Y4_I, W_0_R, W_0_I, 0)
Z1_R, Z1_I, Z5_R, Z5_I = butterfly(Y1_R, Y1_I, Y5_R, Y5_I, W_0_R, W_0_I, 0)
Z2_R, Z2_I, Z6_R, Z6_I = butterfly(Y2_R, Y2_I, Y6_R, Y6_I, W_0_R, W_0_I, 0)
Z3_R, Z3_I, Z7_R, Z7_I = butterfly(Y3_R, Y3_I, Y7_R, Y7_I, W_0_R, W_0_I, 0)
Z8_R, Z8_I, Z12_R, Z12_I = butterfly(Y8_R, Y8_I, Y12_R, Y12_I, W_4_R, W_4_I
                                       , 0)
Z9_R, Z9_I, Z13_R, Z13_I = butterfly(Y9_R, Y9_I, Y13_R, Y13_I, W_4_R, W_4_I
                                       , 0)
Z10_R, Z10_I, Z14_R, Z14_I = butterfly(Y10_R, Y10_I, Y14_R, Y14_I, W_4_R,
                                         W_4_I, 0)
Z11_R, Z11_I, Z15_R, Z15_I = butterfly(Y11_R, Y11_I, Y15_R, Y15_I, W_4_R,
                                         W_4_I, 0)

# Step 3
T0_R, T0_I, T2_R, T2_I = butterfly(Z0_R, Z0_I, Z2_R, Z2_I, W_0_R, W_0_I, 0)
T1_R, T1_I, T3_R, T3_I = butterfly(Z1_R, Z1_I, Z3_R, Z3_I, W_0_R, W_0_I, 0)
T4_R, T4_I, T6_R, T6_I = butterfly(Z4_R, Z4_I, Z6_R, Z6_I, W_4_R, W_4_I, 0)
T5_R, T5_I, T7_R, T7_I = butterfly(Z5_R, Z5_I, Z7_R, Z7_I, W_4_R, W_4_I, 0)

```

```

T8_R, T8_I, T10_R, T10_I = butterfly(Z8_R, Z8_I, Z10_R, Z10_I, W_2_R, W_2_I
                                      , 0)
T9_R, T9_I, T11_R, T11_I = butterfly(Z9_R, Z9_I, Z11_R, Z11_I, W_2_R, W_2_I
                                      , 0)
T12_R, T12_I, T14_R, T14_I = butterfly(Z12_R, Z12_I, Z14_R, Z14_I, W_6_R,
                                         W_6_I, 0)
T13_R, T13_I, T15_R, T15_I = butterfly(Z13_R, Z13_I, Z15_R, Z15_I, W_6_R,
                                         W_6_I, 0)

# Step 4
X0_R_out, X0_I_out, X8_R_out, X8_I_out = butterfly(T0_R, T0_I, T1_R, T1_I,
                                                    W_0_R, W_0_I, 0)
X4_R_out, X4_I_out, X12_R_out, X12_I_out = butterfly(T2_R, T2_I, T3_R, T3_I
                                                    , W_4_R, W_4_I, 0)
X2_R_out, X2_I_out, X10_R_out, X10_I_out = butterfly(T4_R, T4_I, T5_R, T5_I
                                                    , W_2_R, W_2_I, 0)
X6_R_out, X6_I_out, X14_R_out, X14_I_out = butterfly(T6_R, T6_I, T7_R, T7_I
                                                    , W_6_R, W_6_I, 0)
X1_R_out, X1_I_out, X9_R_out, X9_I_out = butterfly(T8_R, T8_I, T9_R, T9_I,
                                                    W_1_R, W_1_I, 0)
X5_R_out, X5_I_out, X13_R_out, X13_I_out = butterfly(T10_R, T10_I, T11_R,
                                                       T11_I, W_5_R, W_5_I, 0)
X3_R_out, X3_I_out, X11_R_out, X11_I_out = butterfly(T12_R, T12_I, T13_R,
                                                       T13_I, W_3_R, W_3_I, 0)
X7_R_out, X7_I_out, X15_R_out, X15_I_out = butterfly(T14_R, T14_I, T15_R,
                                                       T15_I, W_7_R, W_7_I, 0)

#####
print("X0_R_out=" + str(X0_R_out/16))
print("X0_I_out=" + str(X0_I_out/16))
print("X1_R_out=" + str(X1_R_out/16))
print("X1_I_out=" + str(X1_I_out/16))
print("X2_R_out=" + str(X2_R_out/16))
print("X2_I_out=" + str(X2_I_out/16))
print("X3_R_out=" + str(X3_R_out/16))
print("X3_I_out=" + str(X3_I_out/16))
print("X4_R_out=" + str(X4_R_out/16))
print("X4_I_out=" + str(X4_I_out/16))
print("X5_R_out=" + str(X5_R_out/16))
print("X5_I_out=" + str(X5_I_out/16))
print("X6_R_out=" + str(X6_R_out/16))
print("X6_I_out=" + str(X6_I_out/16))
print("X7_R_out=" + str(X7_R_out/16))
print("X7_I_out=" + str(X7_I_out/16))
print("X8_R_out=" + str(X8_R_out/16))
print("X8_I_out=" + str(X8_I_out/16))
print("X9_R_out=" + str(X9_R_out/16))
print("X9_I_out=" + str(X9_I_out/16))
print("X10_R_out=" + str(X10_R_out/16))
print("X10_I_out=" + str(X10_I_out/16))
print("X11_R_out=" + str(X11_R_out/16))
print("X11_I_out=" + str(X11_I_out/16))
print("X12_R_out=" + str(X12_R_out/16))
print("X12_I_out=" + str(X12_I_out/16))
print("X13_R_out=" + str(X13_R_out/16))
print("X13_I_out=" + str(X13_I_out/16))

```

```
print("X14_R_out=" + str(X14_R_out/16))
print("X14_I_out=" + str(X14_I_out/16))
print("X15_R_out=" + str(X15_R_out/16))
print("X15_I_out=" + str(X15_I_out/16))

-----
-- GRUPPO 9: Palma Paolo 306046
--           Porcaro Mario 315888
--           Rinieri Filippo 317894
-----
```